Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleksander Berus

Analiza ogrodja MonoGame za medplatformni razvoj iger na mobilnih platformah

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Matija Marolt

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:
Analiza ogrodja MonoGame za medplatformni razvoj iger na mobilnih platformah

Tematika naloge:
V diplomski nalogi preučite arhitekturo in delovanje ogrodja MonoGame za razvoj medplatformnih mobilnih iger. Predstavite tehnike reševanja različnosti mobilnih platform znotraj skupne programske kode in preučite delovanje cevovoda vsebine. Za demonstracijo razvijte dve igri, 2D in 3D, in preučite potrebne korake, da dosežete enakovredno delovanje iger na več platformah.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleksander Berus, sem avtor diplomskega dela z naslovom:

Analiza ogrodja MonoGame za medplatformni razvoj iger na mobilnih platformah

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 29. oktober 2014 Podpis avtorja:

Za vse nasvete in strokovno pomoč pri izdelavi diplomske naloge se posebno zahvaljujem mentorju doc. dr. Matiji Maroltu. Hvala moji družini in prijateljem za vso podporo in pomoč v času študija.

Kazalo

Seznam uporabljenih kratic in simbolov

ADT (angl. *Android Developer Tools*) - orodja za razvoj aplikacij platforme *Android*

AOT (angl. *Ahead-Of-Time*) - prevajanje pred časom

API (angl. *Application Programming Interface*) - programski vmesnik

ARM (angl. *Advanced RISC Machine*) - napredni RISC stroj

CIL (angl. *Common Intermediate Language*) - skupni vmesni jezik

GLSL (angl. *OpenGL Shading Language*) - OpenGL jezik senčenja

HLSL (angl. *High Level Shader Language*) - visoko-nivojski jezik senčenja

JIT (angl. *Just-In-Time*) - prevajanje ob pravem času

JNI (angl. *Java Native Interface*) - javin domorodni vmesnik

IDE (angl. *Integrated Development Environment*) - integrirano razvojno okolje

OpenGL (angl. *Open Graphics Library*) - odprtokodna grafična knjižnica

OpenGL ES (angl. *OpenGL for Embedded Systems*) - OpenGL za vdelane sisteme

RGBA (angl. *Red Green Blue Alpha*) - rdeča, zelena, modra in prosojnost

SDK (angl. Software Development Kit) - zbirka orodij za razvoj programske opreme

XACT (angl. *Cross-platform Audio Creation Tool*) - medplatformno orodje za izdelavo zvoka

XNA (angl. *XNA's Not Acronymed*) - *XNA* ni kratica

XNI (angl. *XNA for iOS*) - *XNA* za platformo *iOS*

Povzetek

Razvoj računalniških iger je zahteven proces. Ob razvoju iger za mobilne naprave se proces razvoja še nekoliko oteži. Potrebno je razviti isto igro za različne mobilne platforme posebej. Vendar imamo orodja, ki omogočajo medplatformni razvoj iger. Orodja za medplatformni razvoj iger nam omogočajo, da igro napišemo v enem programskem jeziku in jo izvajamo na več različnih mobilnih platformah. V diplomski nalogi podrobno predstavimo arhitekturo in delovanje ogrodja *MonoGame* za medplatformni razvoj mobilnih iger. V delu se osredotočimo na tri najbolj razširjene mobilne platforme – *Android*, *iOS* ter *Windows Phone*. Predstavimo nekaj tehnik reševanja različnosti mobilnih platform znotraj skupne programske kode. V nadaljevanju si ogledamo delovanje *Cevovoda vsebine*, ki olajša delo s teksturami, 3D modeli in ostalo vsebino. Na koncu razvijemo dve (2D in 3D) demonstracijski igri, s katerima prikažemo delovanje ogrodja *MonoGame* na mobilnih platformah *Android* in *Windows Phone*. Ogrodje *MonoGame* olajša razvoj mobilnih iger in s pomočjo ene izvorne kode lahko izdelamo igro za več različnih mobilnih platform.

Ključne besede:

ogrodje MonoGame, medplatformnost, mobilne platforme, mobilne igre

Abstract

The development of computer games is challenging. When developing games for mobile phones, this process is slightly more difficult because we need to develop the same game for several different mobile platforms separately. However, we have tools that allow cross-platform game development. Tools for cross-platform game development enable us to write a game in one programming language and it can be run on multiple mobile platforms. In this thesis, we take a closer look at the architecture and functionalities of *MonoGame* framework for the cross-platform mobile game development. We focus on three major mobile platforms - *Android*, *iOS*, and *Windows Phone* in this paper. We present a number of techniques, how to solve the diversity of mobile platforms within a common codebase. Next we look into the *Content Pipeline* that eases work with textures, 3D models and other content. At the end, we develop two (2D and 3D) demonstrative games to display functionalities of *MonoGame* framework on *Android* and *Windows Phone* mobile platforms. *MonoGame* framework facilitates the development of mobile games and we can create one game that runs on different mobile platforms with the help of one source code.

Keywords:

MonoGame Framework, cross-platform, mobile platforms, mobile games

Poglavje 1

Uvod

1.1 Razvoj računalniških iger

Razvoj računalniških iger je kompleksen, zahteven, dolgotrajen proces. Razvijalec (programer) iger mora poleg dobrega programiranje poznati računalniško grafiko, modeliranje in umetno inteligenco. Poleg programerjev potrebujemo še oblikovalce, tonske mojstre itd. Z namenom, da se olajša in pohitri razvoj iger, so nastala različna orodja: od bolj enostavnejših namenskih programskih knjižnic (angl. library) do ogrodij (angl. framework) ter najbolj zahtevnejših igralnih pogonov (angl. game engine). Izbira orodja je seveda odvisna od naših potreb in kakšen nadzor bi radi imeli na našo programsko kodo (igralni pogoni lahko sami generirajo kodo). Ta orodja imajo že vgrajene rešitve za zgoraj naštetih probleme. Najprej si oglejmo, kakšne so razlike med temi orodji.

1.1.1 Programske knjižnice

Knjižnica je en del programske kode, ki se uporablja pri izdelavi igre. Po navadi vsebuje samo določen tip, nabor funkcionalnosti. To so knjižnice

za računanje fizike in matematike v igrah, knjižnice za zvok, knjižnice za pomoč pri grafičnem upodabljanju ali dostopu do grafične strojne opreme. Knjižnice izoblikujejo skupen del programske kode, ki se večkrat pojavi pri izdelavi igre. Samo s knjižicami ne moramo izdelati celotne igre, potrebujemo osnovo, na kateri lahko gradimo igro. Seveda knjižnice uporabimo z ogrodjem ali igralnim pogonom.

1.1.2 Ogrodje

Ogrodje je zbirka knjižnic in vmesnikov, nad katerimi gradimo igre. Igre vsebujejo tudi kakšna dodatna orodja za lažje rokovanje z ogrodjem. V nasprotju z igralnimi pogoni imamo v tem primeru popolni nazor nad obnašanjem programske kode. Z ogrodji delo poteka na nižjem nivoju, kot delo s pogoni. Če za primer pogledamo ogrodje *MonoGame*, nam ponuja vse potrebno za izgradnjo igre s pomočjo vgrajenih knjižnic in vmesnikov. Vendar moramo sami napisati kodo za izvajanje igre. Z drugimi besedami bi lahko tudi rekli, da je orodje nekakšna vez med vsemi deli programske kode igre.

1.1.3 Igralni pogon

Igralni pogon je polno funkcijo orodje. Z igralnim pogonom poteka razvoj iger v zelo visokem nivoju z uporabo vizualnih razvojnih orodij. Igre kreiramo z razvrščanjem elementov na zaslonu. Namen igralnih pogonov je, da se lahko razvijalci iger bolj osredotočijo na samo vsebino igre in porabijo manj časa za programsko implementacijo igre. Po navadni nam ni potrebno veliko programirati oziroma določene aktivnosti so že v ozadju implementirane, mi se samo odločimo, kako se bodo obnašale. Pogon obvladuje logiko jedra igre, animacije, nalaganje in prikazovanje zaslonov, zaznavanje trkov, fiziko igre ipd. Programiramo samo takrat, kadar hočemo razširit osnovne funkcionalnosti ali spremeniti obnašanje elementov, ki nam jih predpostavlja

igralni pogon in to poteka v večji meri z uporabo skriptnih jezikov. Pogon nam lahko narekuje, kakšen žanr iger lahko z njim ustvarimo. Pri ogrodju nimamo takšne omejitve, edino če bo igra v dva (2D) ali tri (3D) dimenzionalnem svetu.

Lahko se odločimo in nad ogrodjem sami napišemo svoj igralni pogon. V pogonu imamo seznam elementov, katerim določimo položaj na zemljevidu sveta, ogrodje pa nam omogoča izris teh elementov.

1.2 Igre na mobilnih platformah in medplatformnost

V zadnjem času opažamo porast mobilnih naprav ter porast njihovih strojnih in grafičnih zmogljivosti. Razvoj iger se je preselil na mobilne platforme, vendar se tukaj srečujemo z drugačnimi težavami kot pri namiznih igrah. Mobilne naprave imajo, zaradi svoje majhnosti, okrnjeno strojno opremo in predvsem moramo paziti na porabo baterije. Razlikujejo se tudi operacijski sistemi, ki tečejo na teh mobilnih napravah. Pri namiznih napravah se v veliki meri igre razvijajo samo za namizno platformo *Windows*. Pri mobilnih sistemih moramo ciljati na vsaj dve oziroma tri platforme pri razvoju igre. Najbolj pogoste in razširjene platforme so *Android*, *iPhone* in *Windows Phone*. To predstavlja pereč problem, saj če hočemo igro napisati v domorodni kodi, jo moramo za vsako platformo razviti posebej. Za rešitev tega problema moramo uporabit enega od dveh zgoraj naštetih orodij, ogrodje ali igralni pogoni, seveda mora orodje podpirati razvoj medplatformnih iger.

Medplatformne (angl. cross-platform) igre so tiste igre, ki omogočajo, da jih razvijamo samo enkrat in delujejo na več različnih platformah. Pomeni, da si igre delijo ali celo imajo skupno programsko kodo, vendar ko se prevedejo, se prevedejo v domorodni programski jezik ciljne platforme. S tem dosežemo hitro delovanje iger, kot da bi bile v začetku napisane za specifično platformo.

1.3 Cilj diplomskega dela

V diplomski nalogi si bomo ogledali delovanje orodja *MonoGame*, ki omogoča razvoj medplaformnih iger. Osredotočili se bomo na tri najbolj razširjene mobilne platforme *Android*, *iOS (iPhone)* in *Windows Phone*. Ogrodje *MonoGame* smo izbrali zato:

- ker je medplatformno ogrodje;
- je odprtokodno ogrodje, kar pomeni, da si lahko skozi programsko kodo ogledamo delovane in reševanje problemov, ki jih povzroča razvoj iger za mobilne platforme;
- podpira razmeroma enostaven razvoj iger;
- omogoča uporabo obeh glavnih programskih vmesnikov za dostop do grafične strojne opreme – *DirectX* in *OpenGL*;
- ni preveč obširno ogrodje ali igralni pogon, tako da ga lahko podrobno opišemo.

Najprej si bom podrobno ogledali sestavo oziroma arhitekturo ogrodja, nato še s pomočjo igre preverili, če ogrodje pomaga reševati kakšen od vseh omenjenih problemov.

Poglavje 2

Primerjava podobnih ogrodij in pogonov

Ogledali si bomo različna orodja, ki omogočajo medplatformni razvoj mobilnih iger. V tem poglavju bom izpustili primerjavo programskih knjižnic. Same knjižnice ne omogočajo medplatformni razvoj iger, vendar služijo samo kot dodano orodje ogrodjem in igralnim pogonom. V zadnjem času je zelo popularen razvoj iger s pomočjo takih ogrodij. Omejili se bomo samo na bolj znana in izstopajoča ogrodja in pogone. Za medplatformni razvoj mobilnih iger morajo ogrodja in igralni pogoni izpolnjevati naslednje zahteve:

- omogočajo izvajanje iger na mobilnih platformah - *Android*, *iOS (iPhone)* ter *Windows Phone*,
- uporabljajo vsaj enega od programski vmesnik za dostop do grafične strojne opreme – *Direct X* ali *OpenGL*,
- omogočajo uporabo senzorjev in ostalih vhodov na mobilni napravi,
- izvajajo igre, kot domorodna aplikacija.

Vsa spodaj opisana ogrodja in pogon omogočajo uporabo senzorjev in ostalih vhodov na mobilni napravi ter izvajanja oziroma prevajanje igre, kot domorodna aplikacija.

LibGDX

LibGDX [1] je ogrodje za razvoj medplatformnih grafičnih iger. Ogrodje omogoča tako razvoj 2D kot 3D iger. Po funkcionalnosti je zelo podobno ogrodju *MonoGame*.

Ogrodje napisano v programskem jeziku *Java*, vsebuje tudi nekaj komponent napisanih v programskem jeziku *C++*, do katerih lahko dostopamo preko javinega domorodnega vmesnika *JNI* (angl. *Java Native Interface*). Ogrodje je osnovano na programskem vmesniku *OpenGL* za dostop do grafične strojne opreme.

Ogrodje ne podpira razvoja iger za platformo *Windows Phone*, platformi *Android* in *iOS (iPhone)* sta podprti. Pri platformi *iOS (iPhone)* moramo prevedeno javino bajtno kodo (angl. *Java bytecode*) prevesti v izvedljivo kodo *ARM* za platformo *iOS*. To se naredi s pomočjo prevajalnika *RoboVM* [2].

PlayN

PlayN [3] je v programskem jeziku *Java* napisano ogrodje, ki uporablja grafično knjižnico *LWJGL* (*Lightweight Java Game Library*) za dostop do programerskega grafičnega vmesnika *OpenGL*. Zmogljivosti ogrodja so zelo podobna ogrodjema *LibGDX* in *MonoGame*.

Podpri sta mobilni platformi *Android* in *iOS (iPhone)*. Pri izdelavi igre za platformo *iOS* moramo uporabiti prevajalnik *RoboVM*, enako kot pri ogrodju *LibGDX*.

OGRE

OGRE (*Object-Oriented Graphics Rendering Engine*) [4] je grafični pogon oziroma pogon za upodabljanje. Pogon ima objektivno orientiran vmesnik za upodabljanje, ki poenostavi razvoj pri ustvarjanju 3D scen. Ta pogon je nekakšen vmesni korak med ogrodjem in pravim igralnim pogonom. Pogon *ORGE* ima nekaj osnovih funkcionalnosti pogona že vključenih, ostale lahko dodamo z uporabo zunanjih knjižnic.

Pogon je spisan v programskem jeziku *C++*, kar pomeni, da se igre izvajajo v domorodnih tehnologijah na mobilnih platformah. Pogon vsebuje enoten vmesnik za *Direct3D* in *OpenGL*, podpira vse tri mobilne platforme.

Unity

Unity (tudi *Unity3D*) [5] je igralni pogon in razvojno okolje za razvoj medplatformnih iger. Razvojno okolje omogoča enostaven in hiter razvoj iger.

Okolje omogoča, da s pomočjo miške povlečemo elemente na sceno igre ter spreminjamo in nastavljamo njihove lastnosti. Takim igralnim pogonom rečemo *"point and click game engine"*. V vsakem trenutku lahko zaženemo hitri ogled delovanja igre, brez da bi zapustili razvojno okolje. V nasprotju z drugimi orodji, ki tega ne omogočajo, nam ni potrebno prevesti igro in zagnati na napravi. To zelo pohitri razvoj iger. Ko igro želimo prenesti na mobilno napravo, zaženemo dialog za izvoz igre. V dialogu lahko za vsako platformo izberemo ali omogočimo posebne nastavitve, ki bodo pripomogle k boljši igri. Med prevajanjem igre se naredi optimizacija igre za izbrano platformo.

Igralni pogon je napisan v programskem jeziku *C++* in omogoča pisanje preprostih skript v jezikih *C#* ali *JavaScript* (*UnityScript*) [6]. Na mobilni platformi *Windows Phone* pogon uporablja *DirectX* za upodabljanje. Na

platformah *Android* in *iOS (iPhone)* se uporablja *OpenGL* [7].

Poleg omenjenih orodij imamo orodja, ki so neposredna konkurenca ogrodju *MonoGame*, saj vsebujejo iste funkcionalnosti. Opisali bomo dva ogrodja, ki imata korenine v ogrodju *XNA*, katerega naslednik je tudi ogrodje *MonoGame*. Obe ogrodji sta implementaciji ogrodja *XNA 4.0*.

ANX.Framework

ANX.Framework [8] je platformno neodvisno ogrodje za razvoj iger, ki je združljivo z Microsoftovim ogrodjem *XNA*. Pomeni, da je izvorna programska koda pisana za ogrodje *XNA*, deluje brez sprememb z ogrodjem *ANX.Framework*. Pri prenosu iz enaga ogrodja v drugega moramo zamenjati imenski prostor (angl. namespace). `Microsoft.XNA.Framework` preimenujemo v `ANX.Framework`.

Ogrodje deluje samo na platformi *Windows Phone*, ki vsebuje vmesnik *DirectX*.

XNI

XNI [9] je implementacija ogrodja *XNA* verzije 4.0 za mobilno platformo *iOS*. Avtor ogrodja je Matej Jan, ki je bil študent Fakultete za računalništvo in informatiko. Ogrodje je v celoti spisano v programskem jeziku *Objective-C* in trenutno vsebuje osnovne funkcionalnosti ogrodja *XNA*.

Poglavje 3

Zgodovina ogrodij XNA in MonoGame

3.1 Ogrodje XNA

Microsoft XNA je ogrodje za razvoj računalniških iger za platformo *Windows*, igralno konzolo *Xbox 360* in mobilno platformo *Windows Phone 7*. Izdano je bila z namenom, da poenostavi neodvisnim razvijalcem (angl. indie developer) razvoj iger. *XNA* ogrodje je zasnovano na ogrodju *.NET* ter v ozadju komunicira z grafično kartico preko programskega vmesnika *Direct3D* za dostop do grafične strojne opreme.

Leta 2006 je bila izdana prva verzija ogrodja skupaj z različnimi orodji imenovana *XNA Game Studio Express*. Nato so si sledile verzije *XNA Game Studio 2.0* (izdana leta 2007), *XNA Game Studio 3.0* (izdana 2008), *XNA Game Studio 3.1* (2009) ter zadnji verziji *XNA Game Studio 4.0* (2010) in *XNA Game Studio 4.0 Refresh* (2011). Od 31. januarja 2013 podjetje Microsoft aktivno več ne razvija ogrodja *XNA* [10].

3.2 Ogrodje MonoGame

Ogrodje *MonoGame* je odprtokodna, brezplačna implementacija Microsoftovega ogrodja *XNA* verzije 4.0 [11]. Cilj je omogočiti enostaven prenos *XNA* iger pisanih za konzolo *Xbox 360*, platformo *Windows* in *Windows Phone* ter tudi na druge platforme. Trenutno podprte platforme so *iOS*, *Android*, *MacOS X*, *Linux*, *Windows 8 RT* ter igralni konzoli *Ouya* in *Play Station Mobile* [11].

V začetku se je ogrodje *MonoGame* imenovalo *XNATouch* in cilj tega ogrodja je bilo sprva omogočiti *XNA* razvijalcem prevedbo svojih iger za platformo *iPhone*. Podpora je bila samo za 2D *XNA* igre. Ogrodje *MonoGame* je uporabilo veliko delov knjižnic *SilverSprite* in *Mono.XNA* za svojo osnovo. December 2009 je izšla prva verzija ogrodja, ki je podpirala samo platformo naprav *iPhone*.

Velik korak so naredili razvijalci ogrodja z izdajo verzije *3.0*. Dodana je bila podpora za 3D programski vmesnik (*API*) (angl. *Application Programming Interface*) in funkcionalnosti, ki jih prinaša *OpenGL 2.0*. Prejšnje verzije so temeljile na *OpenGL ES 1.x*. Dodana je bila podpora za operacijske sisteme *Android*, *Mac OS*, *Linux* in *Windows* (z *OpenGL*).

S prihodom operacijskih sistemov *Windows 8* in *Windows Phone 8* je podjetje Microsoft ukinilo razvoj ogrodja *XNA*[10, 12]. Tistega leta sta bila dodane še te dve platformi k ogrodju *MonoGame* verzije *3.0*. Z razliko od prejšnjih verzij, ki so temeljile na *OpenGL*, zdaj ogrodje omogoča tudi komunikacijo z grafično kartico preko vmesnika *DirectX* pri *Windows* platformah.

Trenutna verzija (*3.2*) ogrodja v večini podpira *XNA* vmesnik verzije *4.0* in omogoča enostaven prenos 3D *XNA* iger na vse zgoraj opisane sisteme.

Razvoj ogrodja *MonoGame* je še vedno v teku. V najnovejših verzijah je bilo odpravljenih veliko napak. Nekatere manj pogosto uporabljene funkcionalnosti iz ogrodja *XNA* še vedno manjkajo in bodo sčasoma implementirane.

Poteka še tudi razvoj za platformi *ANGLE* in *PlayStation 4* [11, 13].

Poleg samega ogrodja *MonoGame*, ki skrbi za izrisovanje slike na zaslone mobilne naprave, je orodje z imenom *Cevovod vsebine* (angl. *Content Pipeline*) tudi pomemben sestavni del ogrodja. Trenutno je *Cevovod vsebine* še v zgodnjih fazah razvoja. Primarni namen je bil najprej dostavi ogrodje *MonoGame*. To praznino seveda lahko zapolnimo z uporabo *Cevovoda vsebine* ogrodja *XNA*, saj ogrodje *MonoGame* pooseblja vse funkcionalnosti ogrodja *XNA* in tako lahko uporabimo vsebino iz slednjega *Cevovoda vsebine*.

Poglavje 4

Razvojno okolje in orodja

Razvoj iger z orodjem *MonoGame* poteka v jeziku *C#* in orodju *Visual Studio 2012* z ustreznimi razširitvami, kot so: *Windows Phone 8 SDK*, *Android SDK for Windows* in *Xamarin*.

4.1 Visual Studio 2012

Visual Studio 2012 [14] je integrirano razvojno okolje (*IDE*) (angl. *Integrated Development Environment*), ki je zbirka orodij in storitev za razvoj namiznih, mobilnih, spletnih aplikacij. Omogoča razvoj v programskih jezikih *.NET* okolja (*C#* in *Visual Basic*), *HTML/JavaScript* ter v jeziku *C++*. Visual Studio je zelo napredno programsko orodje, ki olajša izdelavo programske opreme (aplikacij in iger). Urejevalnik kode podpira označevanje sintakse, samodejno vstavljanje, zlaganje in preoblikovanje kode. Razhroščevanje kode nam omogoča postavitev ustavitvenih točk (angl. brakepoints), ogled vrednosti v spremenljivkah ter dostop do sklada klicev v času izvajanja kode. Poleg pisanja programske opreme za zgoraj omenjene aplikacije, lahko razvijamo tudi programe za grafično kartico - senčilnikov (angl. shaders) v jeziku *HLSL*. Orodje s takim naborom funkcionalnosti je dostopno v plačljivi verziji.

Poleg plačljivih različic obstajajo tudi brezplačne [15], ki ne vsebujejo vseh funkcionalnosti plačljivih različic. Orodje razvijajo pri podjetju Microsoft.

4.2 Windows Phone 8 SDK

Windows Phone 8 SDK je brezplačna zbirka orodij, ki doda *Visual Studio* zmožnost razvijanja programske opreme za mobilno platformo *Windows Phone*. Zbirka orodij vsebuje vse dodatne mobilne knjižnice, ki ni so del standardnega *.NET* ogrodja, emulator z razhroščevalnikom ter orodja za povezovanje z fizično napravo.

4.3 XNA Game Studio Extension

XNA Game Studio Extension je dodatek (angl. extension) za *Visual Studio 2012*. Z ukinitvijo ogrodja *XNA* se je tudi ukinila podpora ogrodja *XNA* v *Visual Studiu 2012*. Ta dodatek omogoča odpiranje in urejanje projektov ogrodja *XNA* ter pogajanja *Cevovoda vsebine* ogrodja *XNA* v *Visual Studiu 2012*.

4.4 Android SDK for Windows (ADT - Android Developer Tools)

Android SDK je zbirka orodij, katera nam mogoči razvoj mobilnih aplikacij za platformo *Android*. Sestavljata ga dva pomembna orodja *SDK Manager* in *AVD Manager*. Prvo orodje nam mogoča dodajanje različnih gradnikov tega orodja, kot so emulator, orodja za prevajanje kode, prenašanje novejših verzij tega orodja itd. Namen *AVD Managerja* je upravljanje z navideznimi napravami - emulatorji platforme *Android*. Podprta sta jezika *Java* in *C++*

za razvoj aplikacij in iger. *Andorid SDK* razvija podjetje Google in je brezplačno.

4.5 Xamarin

Xamarin [16] je zbirka orodij, ki omogoča razvoj domorodnih aplikacij za platformi *Androdi* in *iOS (iPhone)* v programskem jeziku *C#*. Vsebuje vrsto knjižnic in vmesnikov, ki omogočajo dostop do knjižnic na domorodnih platformah. Na platformi *Android* orodje razširi osnovne *Javnine* knjižnice s knjižnicami napisanimi v *.NET* ogrodju. Poleg programskega jezika *Java* lahko programiramo *Android* aplikacije tudi v jeziku *C#*. Podobno velja tudi za *iOS (iPhone)* platformo, kjer so domorodne knjižnice v *Objective-C* programskemu jeziku razširjene s knjižnicami v *.NET* upravljalni kodi (angl. *Managed Code*). Poleg samih knjižnic nam mogoča razvoj, prevajanje in razhroščevanje *androidnih* in *iOS* aplikacij v okolju *Visual Studio*. Orodje *Xamarin* ni brezplačno in ga je potrebno za vsako platformo *Xamarin.Android* in *Xamarin.iOS* kupiti posebej. Brezplačna preizkusna različica je omejena na 30 dnevno uporabo, zaganjanje aplikacije samo na emulatorju ter končno velikost razvojnega projekta.

4.6 Git, GitHub in GitHub for Windows

Ogrodje *MonoGame* je pretežno še v razvoju. Zato je zelo pomembno, da imamo dostop do najnovejših verzij izvorne kode. Izvorna koda ogrodja *MonoGame* je dostopna na spletnem repozitorju *GitHub*.

GitHub je spletno gostovanje storitev za razvoj programskih rešitev. Bazirano je na sistemu za nadzor revizij izvorne kode *Git*. *GitHub* nam omogoča hraniti različne verzije izvorne kode ter usklajevanje med njimi.

Orodje *GitHub for Windows* omogoča prenos izvorne kode iz spletnega repozitorja *GitHub* na lokalni računalnik.

Poglavje 5

Zgradba ogrodja MonoGame

Kot smo že omenili, je ogrodje *MonoGame* odprtokodna implementacija ogrodja *XNA* oziroma implementacija njegovih imenskih prostorov (angl. namespace) in razredov (angl. class). Ogrodje *MonoGame* omogoča *XNA* razvijalcem čim večjo prenosnost kode med različnimi platformami s čim manjšim naporom. Moto ogrodja *MonoGame* je *"Napisati enkrat, igrati vsepovsod"* (angl. *"Write Once, Play Everywhere"*).

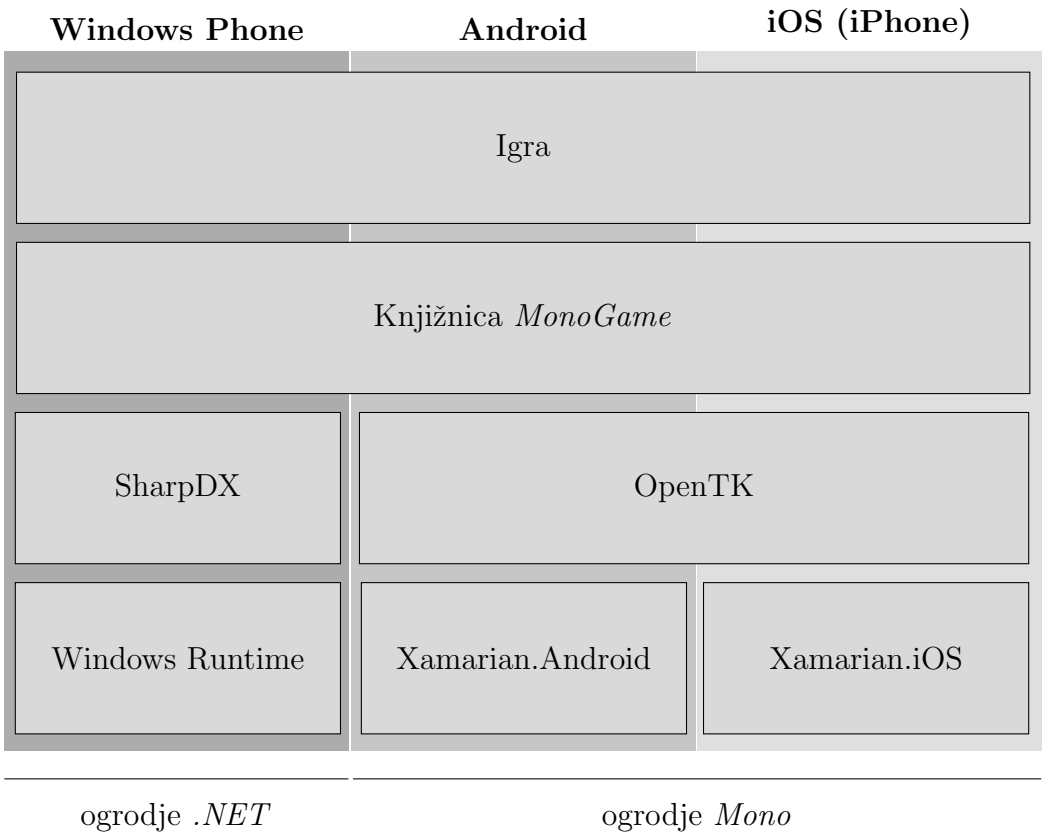
5.1 Arhitektura ogrodja MonoGame

Igre napisane z ogrodjem *MonoGame* imajo 4 nivojsko arhitekturo (Slika 5.1).

5.1.1 Programska koda igre (nivo 4)

Na najvišjem nivoju je *izvorna koda same igre*, *vsebina* (teksture, modeli, zvok,...) ter *pomožne programske knjižnice*, ki so pisane za *.NET* okolje.

Igre z ogrodjem *MonoGame* so napisane samo v programskem jeziku *C#*, za razliko od okolja *.NET*, kjer lahko pišemo programe tudi v drugih program-



Slika 5.1: Shema 4 nivojske arhitekture ogrodja *MonoGame*.

skih jezikih (*Visual Basic.NET* in *F#*). Pri pomožnih knjižnicah imamo isto omejitev glede izbire programskega jezika. Za platformi *Android* in *iOS* je podjetje Xamarin razvilo prevajalnik samo za programski jezik *C#* [16]. Seveda pri platformi *Windows Phone* lahko uporabimo poljubni programski jezik okolja *.NET*, vendar nam to zmanjša prenosljivost kode med različnimi platformami.

Vsebina (angl. *Content*) je poleg same programske kode tudi zelo pomemben sestavni del današnjih iger. Vsebino predstavljajo gradniki, kot so teksture in modeli. Te gradnike uvozimo iz drugih programov in pretvorimo v obliko, ki jo ogrodje *MonoGame* zna prebrati. Za pretvorbo poskrbi *Cevovod*

vsebine (angl. *Content Pipeline*). Vse datoteke vsebine imajo po pretvorbi datotečno končnico `.xnb` (ne glede na vrsto vsebine) in se nahaja v mapi `Content`. Pri vsebini moramo paziti v kakšen format jo pretvorimo, ker vse platforme ne podpirajo vseh formatov. Več o tem in *Cevovodu vsebine* si bomo podobno ogledali v 6. poglavju.

5.1.2 Knjižnica MonoGame (nivo 3)

Nivo nižje se nahajajo knjižnice ogrodja *MonoGame*. Za vsako platformo posebej imamo po dve knjižnic:

- *MonoGame.Framework*
- *MonoGame.Framework.Net*

Do verzije 3.2 je bila samo ena glavna knjižnica *MonoGame.Framework*. Nato se je del kode, ki je namenjen povezovanju z omrežjem, ločil od glavne knjižnice. Ločitev je nastala z razlogom, da se odstrani neposredna odvisnost od zunanje knjižnice *Lidgren.Network*. Knjižnica *Lidgren.Network* nam omogoča uporabo mrežnih protokolov *UDP* in *TCP* v igri. Večina iger ne potrebuje te funkcionalnosti. Z ločitvijo programske kode se je tudi manjšala velikost same glavne knjižnice *MonoGame.Framework*. Pri mobilnih platformah je zelo pomembna velikost aplikacij, saj smo zelo omejeni s pomnilniškim prostorom. Zdaj lahko po potrebi vključimo knjižnico *MonoGame.Framework.Net*.

Na tem nivoju ogrodje MonoGame predstavlja abstrakcijo nad različnimi programski vmesniki do grafične strojne opreme (*OpenGL* in *DirectX*) ter različnimi ciljnimi platformami (*Android*, *iOS*, *Windows Phone* ...). Če se vkopljemo v izvorno kodo, opazimo, da je del kode med posameznimi platformami skupen. To so predvsem funkcionalnosti, ki niso neposredno

vezane na specifičnost platform. Kot primer lahko vzamemo razred `Vector2` v imenskem prostoru `Microsoft.Xna.Framework`.

Drugi del izvorne kode je vezan na posamezno platformo. Ob prevajanju moramo poskrbeti, da se prevede samo tisti del kode, katere funkcionalnosti platforma omogoča. Ta problem se lahko reši na več načinov. Specifičen del izvorne kode lahko ovijemo z `#pragmo` direktive prevajalnika (angl. compiler directive). Če pogoj ni resničen (ciljna platforma ni izbrana), prevajalnik izloči tisti del kode.

```
namespace Microsoft.Xna.Framework.Media
{
    public sealed class Album : IDisposable
    {
#if WINDOWS_PHONE
        private MsAlbum album;
#else
        private Artist artist;
        private Genre genre;
        private string album;
        private SongCollection songCollection;
#if WINDOWS_STOREAPP
        private StorageItemThumbnail thumbnail;
#elif IOS
         private MPMediaItemArtwork thumbnail;
#elif ANDROID
        private Android.Net.Uri thumbnail;
#endif
#endif
```

Primer kode 5.1: Primer uporabe `#pragme` (izbrana platforma je Android).

Drugi način je, da imamo več vzporednih datotek in delne razrede (angl. `partial class`). Skupaj sestavljajo razred, ki cilja na specifično platformo. Za razred `Texture2D` imamo osnovno datoteko `Texture2D.cs`, ter za vsako platformo svojo datoteko `Texture2D.DirectX.cs` ali `Texture2D.OpenGL.cs`. Osnovna datoteka in ena platformno specifična datoteka tvori končni razred `Texture2D`. Ta način je uporabljen takrat, kadar je velik del programske kode vezan oziroma različen za posamezno platformo. To lahko povzroči veliko nepreglednost v programski kodi, če bi bilo vse v eni datoteki.

```
public partial class Texture2D : Texture
{
    //...
    protected Texture2D(GraphicsDevice graphicsDevice, int width,
        int height, bool mipmap, SurfaceFormat format, SurfaceType
        type, bool shared)
    {
        if (graphicsDevice == null)
            throw new ArgumentNullException("Graphics Device...");
        this.GraphicsDevice = graphicsDevice;
        this.width = width;
        this.height = height;
        this._format = format;
        this._levelCount = mipmap ? CalculateMipLevels(width,
            height) : 1;
        if (type == SurfaceType.SwapChainRenderTarget) return;
        PlatformConstruct(width, height, mipmap, format, type,
            shared);
     }
    //...
}
```

Primer kode 5.2: Delni razred `Texture2D` v osnovni datoteki `Texture2D.cs`.

```
public partial class Texture2D : Texture
{
    private bool _shared;
    private bool _renderTarget;
    private bool _mipmap;
    private void PlatformConstruct(int width, int height, bool
        mipmap, SurfaceFormat format, SurfaceType type, bool shared)
    {
        _shared = shared;
        _renderTarget = (type == SurfaceType.RenderTarget);
        _mipmap = mipmap;

        // Create texture
        GetTexture();
        //...
    }
}
```

Primer kode 5.3: Delni razred `Texture2D` v datoteki `Texture2D.DirectX.cs` za platformo *Windows Phone*.

```
public partial class Texture2D : Texture
{
    private void PlatformConstruct(int width, int height, bool
        mipmap, SurfaceFormat format, SurfaceType type, bool shared)
    {
        this.glTarget = TextureTarget.Texture2D;
        //...
    }
}
```

Primer kode 5.4: Delni razred `Texture2D` v datoteki `Texture2D.OpenGL.cs` za platformi *Android* in *iOS*.

Ker je ogrodje *MonoGame* nastalo po vzoru ogrodja *XNA*, ogrodje *MonoGame* ohranja enako hierarhijo in strukturo razredov. Kar pomeni, da so uporabljeni identični imenski prostori kot v ogrodju *XNA* [17, 18] in teoretično omogoča prenos iger pisanih za ogrodje *XNA*. Ogrodje *MonoGame* vsebuje naslednje imenske prostore (angl. namespace):

- `Microsoft.Xna.Framework`
- `Microsoft.Xna.Framework.Audio`
- `Microsoft.Xna.Framework.Content`
- `Microsoft.Xna.Framework.Graphics`
- `Microsoft.Xna.Framework.Graphics.PackedVector`
- `Microsoft.Xna.Framework.Input`
- `Microsoft.Xna.Framework.Input.Touch`
- `Microsoft.Xna.Framework.Media`
- `Microsoft.Xna.Framework.Net`
- `Microsoft.Xna.Framework.Storage`

5.1.3 Knjižnice za komunikacijo s strojno opremo (nivo 2)

V drugem nivoju so knjižnice, ki omogočajo komunikacijo s strojno opremo. Tu imamo dve možnosti. Če mobilna platforma omogoča dostop preko programskega vmesnika *DirectX* do grafične strojne opreme se uporablja knjižnica *SharpDX* (*Windows Phone*). V primeru, da imamo opravka z *OpenGL* (*Android* in *iPhone*) se v tem nivoju uporablja knjižnica *OpenTK*. Tu imamo tudi že prej omenjeno knjižnico *Lidgren.Network* za komunikacijo z omrežjem. Posamezne knjižnice bomo bolj nadrobno opisali v naslednjih razdelkih tega poglavja.

5.1.4 Izvajalno okolje (nivo 1)

Spodnji nivo sestavlja izvajalno okolje (angl. runtime environment) in programski vmesniki do grafične strojne opreme. V tem nivoju imamo direktni dostop do programskega vmesnika do grafične strojne opreme. Do vmesnika raje dostopamo nivo višje preko knjižnic *SharpDX* ali *OpenTK*. Izvajalno okolje na platform *Windows Phone* je ogrodje *.NET*. Na platformah *Android* in *iPhone* je ogrodje *Mono*.

Orodje *.NET* omogoča izvajanje programov pisanih v programskem jeziku *C#*. Za mobilne sisteme *Windows Phone* imamo posebno verzijo tega okolja imenovano *Windows Runtime* (tudi *WinRT*). To izvajano okolje teče znotraj varnostno omejenega okolja (angl. sandboxed environment), kar omogoča večjo varnost in zanesljivost aplikacij oziroma iger.

Na platformah, ki niso tipa *Windows*, je ogrodje *Mono* zamenjava za ogrodja *.NET*. *Mono* je odprtokodna implementacija ogrodja *.NET*. Za mobilne platforme je podjetje Xamarin razvilo dve posebni izvedbi tega okolja: za platformo *Android* je *Xamarin.Android* in za *iOS (iPhone)* je *Xamarin.iOS* [16, 19]. Implementaciji se med seboj razlikujeta. Na platformi *Android* prevajalnik prevede izvorno kodo v skupni vmesni jezik (*CIL*) (angl. *Common Intermediate Language*), ki se nato ob zagonu igre ponovno prevede ob pravem času (*JIT*) (angl. *just-in-time*) v domorodni izvajalni jezik. Ta način je enak kot v orodju *.NET*. Do domorodnih funkcionalnosti sistema *Android* dostopamo preko Javinega domorodnega vmesnika *JNI* (angl. *Java Native Interface*). Platforma *iOS* (*iPhone*) ne omogoča prevajanja v času (*JIT*), zato je podjetje Xamarin razvilo lastni prevajalnik [16, 19]. Namenski prevajalnik prevede pred časom (*AOT*) (angl. *ahead-of-time*) izvorno kodo jezika *C#* v domorodni zbirni jezik *ARM*. V obeh primerih se aplikacije oziroma igre izvajajo kot domorodne aplikacije.

5.2 SharpDX

SharpDX je odprtokodni, platformsko neodvisni vmesnik *DirectX* za okolje *.NET* [20]. Sami razredi so neposredno, samodejno ustvarjeni iz glavnih datotek *DirectX C++* (datoteke z končnico `.h` v jeziku *C++*) in so zgolj navezava na *C++* knjižnice vmesnika *DirectX*. Celotna knjižnica *SharpDX* je spisana v jeziku *C#* in omogoča razvoj visoko zmogljivih iger v 2D in 3D upodabljanju z zvokom v realnem času. Kljub temu, da dostopamo do strojne opreme z upravljalno kodo (angl. managed code), je izvajanje knjižnice hitro in ni vidnih vplivov na hitrost izvajanja [21]. S to knjižnico lahko dostopamo do vrsto nizko-nivojskih vmesnikov za dostop do grafične in zvočne strojne opreme:

- *Direct3D* - verzije 9, 10.0, 10.1, 11, 11.1, 11.2
- *DirectCompute*
- *Direct2D* - verziji 1.0 in 1.1
- *DirectWrite*
- *D3DCompiler*
- *DXGI* - verzije 1.0, 1.1, 1.2
- *DirectSound 8*, *XAudio2*, *XAPO*, *XACT3*, *X3DAudio*
- *DirectInput 8*, *XInput* in *RawInput*

SharpDX je uporabljen v ogrodju *MonoGame* pri mobilnih platformah, ki uporabljajo za izrisovanje *DirectX*. V našem primeru je to pri mobilni platformi *Windows Phone*.

5.2.1 Windows Phone

Na platformi *Windows Phone* imamo dostop do grafične strojne opreme preko programskega vmesnika *DirectX* verzije 11. Platforma ima programirni grafični cevovod in omogoča izvajanje programov neposredno na grafičnem procesorju. Take programe imenujemo senčilniki (angl. shader). Nabor ukazov v vmesniku *DirectX 11* ima omejen nivo zmogljivosti (angl. feature level) na nivo `9_3` [22, 23]. To pomeni, da lahko uporabljamo programski jezik modela senčilnikov (angl. shader model) verzije 4.0, vendar smo omejeni na funkcionalnosti iz verzije 2.0 [23, 24]. Jezik senčilnikov za vmesnik *DirectX* je visoko-nivojski jezik senčenja *HLSL* (angl. *High Level Shading Language*). Več o grafičnem cevovodu in jeziku *HLSL* bomo pisali v naslednjem razdelku o *Cevovodu vsebine*.

5.3 OpenTK

OpenTK [25] (tudi *Open Toolkit*) je nizko-nivojska knjižnica za dostop do vmesnikov *OpenGL*, *OpenCL* in *OpenAL* na grafični kartici. Primerna je za razvoj računalniških iger in znanstvenih aplikacij in vseh ostalih projektov, ki zahtevajo 3D grafiko, zvok ali računske zmožnosti (angl. compute functionality).

Za mobilne platforme je namenjena posebna oziroma okrnjena verzija *OpenGL ES* (*OpenGL for Embedded Systems – OpenGL za vdelane sistme*). Ker je *OpenGL ES* podmnožica naborov ukazov *OpenGL*-a [26], lahko *OpenTK* uporabimo tudi na platformah kot so *Android* in *iOS* (*iPhone*). Orodje *MonoGame* pridoma izkorišča funkcionalnosti knjižnice *OpenTK* za omenjeni dve platformi.

5.3.1 Android in iOS (iPhone)

Med platformami *Android* in *iOS* (*iPhone*) ni večjih razlik pri dostopanju do grafične strojne opreme z programskim vmesnikom *OpenGL ES*. Manjše razlike med platformama se pojavijo edino pri implementacji knjižnice *OpenTK*. Obe platformi podpirata zadnji dve verziji vmesnika *OpenGL*, tako verzijo 2.0 kot tudi 3.0 [27, 28]. Obe platformi imata programirni grafični cevovod. Seveda lahko uporabljamo tudi *OpenGL ES* verzije 1.x, vendar ta verzija uporablja fiksni grafični cevovod in ne omogoča izvajanje senčilnikov. Ogrodje *MonoGame* za izvajanje potrebuje programirni grafični cevovod. Programski jezik senčilnikov pri vmesniku *OpenGL* je visoko-nivojski jezik senčenja *GLSL* (angl. *OpenGL Shading Language*). Senčilnikov v jeziku *GLSL* nam ni potrebno pisati, saj *MonoGame* vsebuje orodje za pretvorbo senčilnikov iz jezika *HLSL* v jezik *GLSL*. Več o pretvorbi senčilnikov si bomo bolj natančno ogledali v naslednjem poglavju o *Cevovodu vsebine*.

5.4 Lidgren.Network

Lidgren.Network je mrežna knjižnica za ogrodje *.NET*, ki vsebuje vmesnik za povezovanje med odjemalcem in strežnikom, branje ter pošiljanje sporočil preko vtičnice *UDP* [29]. Dodatne lastnosti te knjižnice so pošiljanje nepovezanih sporočil (naslovnik je znan, vendar ni stalne povezave), šifriranje sporočil, simulacije slabega omrežja oziroma povezave.

Razredi orodja *MonoGame* v imenskem prostoru `MonoGame.Framework.Net` uporabljajo knjižnico *Lidgren.Network* za pisanje omrežnih, večigralskih (angl. multiplayer) iger ter vodenje globalnih lestvici najboljših rezultatov (angl. highscores leaderboards).

Poglavje 6

Cevovod vsebine

Vsebina (angl. *content*) je poleg same programske kode pomemben del današnjih iger. Dostikrat se sploh ne zavedamo, da moramo najprej izdelati vsebino, nato se posvetimo programiranju. Vsebino predstavljajo gradniki, kot so *teksture*, *slike*, *3D modeli*, *grafični učinki - senčilniki*, *glasba*, *zvočni učinki*, *pisave* ter vse ostalo, kar pripomore izboljšati uporabniško izkušnjo. Danes si igre brez vsebine težko predstavljamo. Seveda vsebino lahko ustavimo s pomočjo kode. Vendar je bolj učinkovita izbira izdelava vsebina z namenskimi programi kot so *Autodesk 3ds Max*, *Blender*, *Adobe Photoshop*, *ACID Pro*, *Adobe Soundbooth* itd. Toda taka vsebina postane lahko zelo problematična. Imamo zelo veliko različnih orodij za izdelavo vsebine, ki shranjujejo vsebino v svoje specifične formate. Ogrodje *MonoGame* podpira samo določene formate vsebine. Za pretvorbo in obdelavo v pravilno obliko med različnimi formati bi potrebovali različna orodja ter skripte za uvoz in izvoz. Orodje *Cevovod vsebine* (angl. *Content Pipeline*) nam te stvari zelo poenostavi. Poleg same pretvorbe vsebine v ustrezne formate, vsebino zapiše v format *.NET* in pripravi vsebino že pred samim izvajanjem igre. S tem pospeši uvoz vsebine v igro in posledično pohitri zagon igre.

Optimalno obdelana vsebina (velikost datoteke in hitrost nalaganje) ima

zelo velik pomen pri mobilnih napravah, ker smo zelo omejeni s pomnilnikov (tako začasni spomnim kot pomnilnik za shranjevanje). Zaradi omejitve z glavnim pomnilnikom (začasnim spominom), mobilni operacijski sistemi odstranijo vso vsebino iz pomnilnika grafične kartice in del iz glavnega pomnilnika, ko aplikacija oziroma igra preide v ozadje (odpremo drugo aplikacijo ali sprejmemo klic). Ko ponovno igra preide nazaj v ospredje, je potrebo ponovno naložiti vsebino, ki je bila odstranjena. Po drugi strani smo tudi omejeni s samo velikostjo aplikacije oz. igre, ki jo lahko naložimo na mobilno napravo. Posledica te pohitritve je tudi daljša življenjska doba baterije.

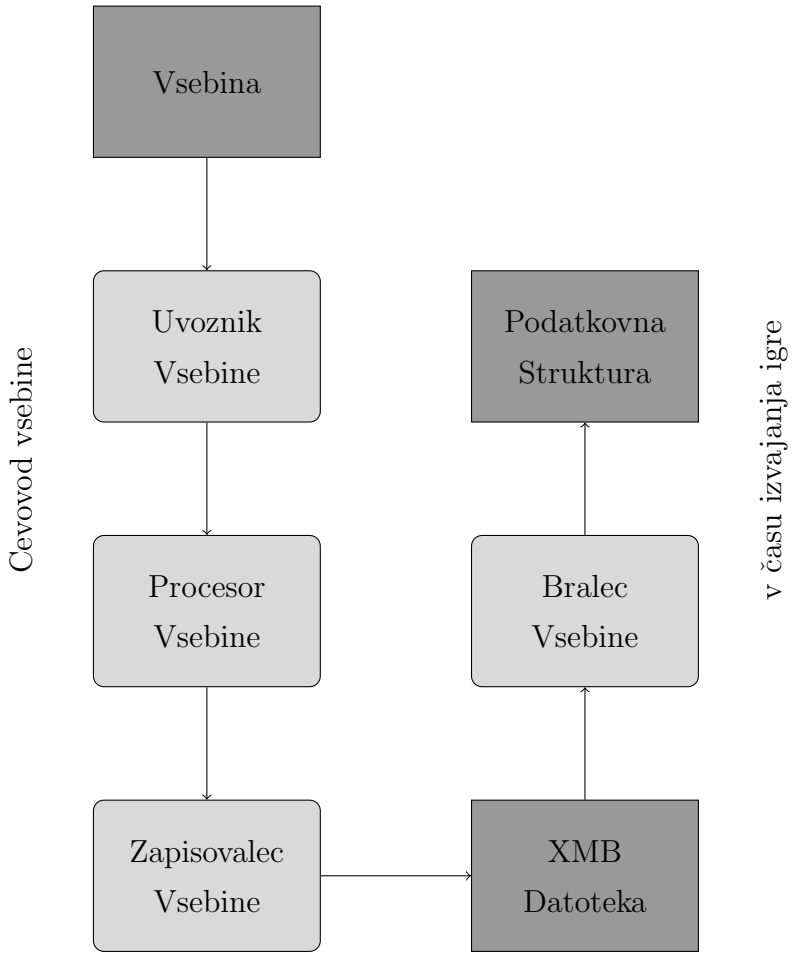
Največje razlike pri razvoju medplatformnih igrer se pojavijo prav v vsebini. Za različne platforme je potrebno pripraviti določen format iste vsebine. Pri sami programski kodi prevajalnik poskrbi, da se izbere določen del kode, ki je namenjen posamezni platformi. Za vsebino mora razvijalec poskrbeti, da ročno doda končnemu projektu ustrezen format vsebine. Razlike v podpori formatov se predvsem pojavijo zaradi različne strojene opreme, različnih mobilnih operacijskih sistemov ali zaradi same implementacije ogrodja *MonoGame*. Kljub temu, bomo videli, da orodje *Cevovod vsebine* močno olajša in poenostavi uvoz vsebine. V nadaljevanju si bomo podrobno ogledali za vsak tip vsebine, kako poteka pretvorba in uvoz.

6.1 Delovanje Cevovoda vsebine

Predem se dotaknemo, kako izgleda uvoz posameznega tipa oziroma formata vsebine, si bomo naprej ogledali v splošnem delovanje *Cevovoda vsebine*.

Vsaka vsebina, ki jo želimo uvoziti in pretvoriti, mora iti čez štiri ključne korake [30]. Ti koraki so prikazani na Sliki 6.1.

Na začetku, v prvem koraku preberemo in uvozimo vsebino (tudi angl. asset) s pomočjo *uvoznika vsebine* (angl. *Content Importer*). Glavni namen uvoznika vsebine je branje podatkov iz datoteke, zato da lahko vsebino



Slika 6.1: *Cevovod Vsebine* s koraki [30].

kasneje v *Cevovodu vsebine* oblikujemo. Izhod tega koraka je objekt (angl. object), ki ga lahko v naslednjem koraku *Cevovod vsebine* obdela.

Drug korak je *procesor vsebine* (angl. *Content Processor*). *Procesor vsebine* sprejeme podatke od *uvoznika vsebine* v surovi obliki (angl. raw format) in vrne obdelano vsebino. V tem koraku pripravimo vsebino, izvajamo različne manipulacije nad vsebino, dodamo podatke ali odstranimo nepotrebne podatke, sestavimo več različnih vsebin v eno vsebino itd. Ta korak je najpomembnejši del celega cevovoda. Končni rezultat je podatkovna

struktura obdelanih podatkov.

V tretjem koraku, na koncu *Cevovoda vsebine*, pridobljene podatke shranimo v binarno datoteko. To naredimo s pomočjo *zapisovalca vsebine* (angl. *Content Writer*). Ta vzame obdelano vsebino iz prejšnjega koraka (vsebina, ki je prišla iz procesorja vsebine) in jo zapiše v binarno datoteko s končnico `.xmb`. To datoteko imenujemo tudi vmesni format oz. vmesna datoteka. V resnici igra potrebuje podatke v obliki iz prejšnjega korak, vendar, da lahko vsebino prenesemo v igro, jo moramo najprej shrani v vmesno datoteko.

Zadnji, četrti korak poteka v času izvajanja igre (angl. Game Runtime). Vsebino naložimo s pomočjo *nalagalnika vsebine* (angl. *Content Loader*). V tem koraku naredimo obratni proces, kot smo ga naredi korak prej. *Bralec vsebine* (angl. *Content Reader*) vsebino prebere iz vmesne datoteke in pretvori v podatkovno strukturo, ki jo ogrodje zna prebrati. V igri vsebino naložimo z metodo:

```
T asset = Content.Load<T>("asset"); //asset.xmb
```

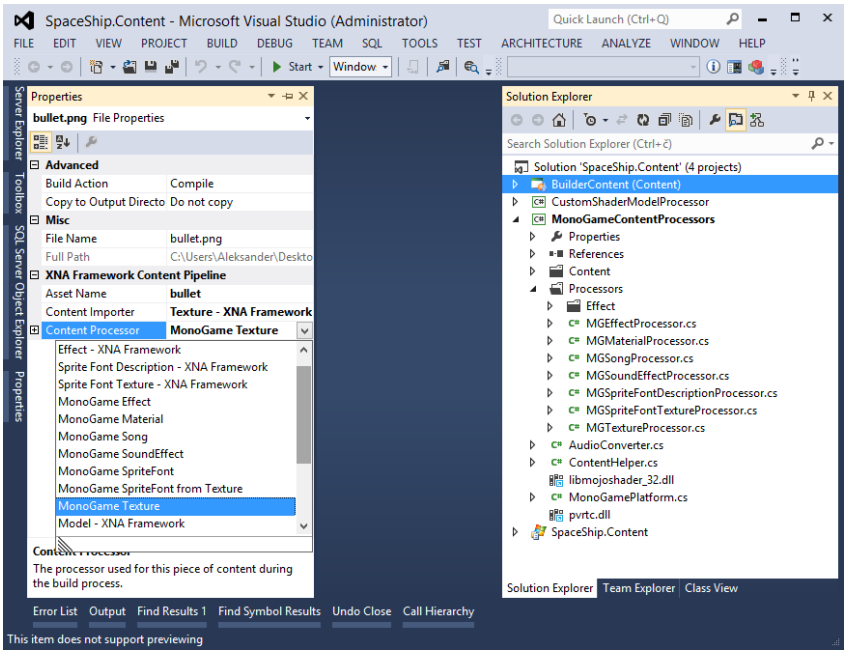
Primer kode 6.1: Medtoda za nalaganje vsebine v igro, razred `T` predstavlja generični razred tipa vsebine

V prvem in drugem koraku smo omenili *uvoznika vsebine* (angl. *Content Importer*) in *procesor vsebine* (angl. *Content Processor*). Obadva sodelujeta oziroma sta odvisna med seboj. Za primer vzemimo teksturo ali sliko, ki jo želimo uporabiti v igri. Za uvoz teksture uporabimo *uvoznik tekstur* (angl. *TextureImporter*) in nato sliko obdelamo s pomočjo *procesorja tekstur* (angl. *TextureProcessor*). *Cevovod vsebino* ima kar zajetno število pripravljenih uvoznikov vsebine (11 uvoznikov) in procesorjev vsebine (9 procesorjev). Seveda je *Cevovod vsebine* razširljiv in omogoča pisanje novih uvoznikov in procesorjev. S tem lahko razširimo cevovod in omogočimo uvoz za še večje število različnih formatov vsebine v igro.

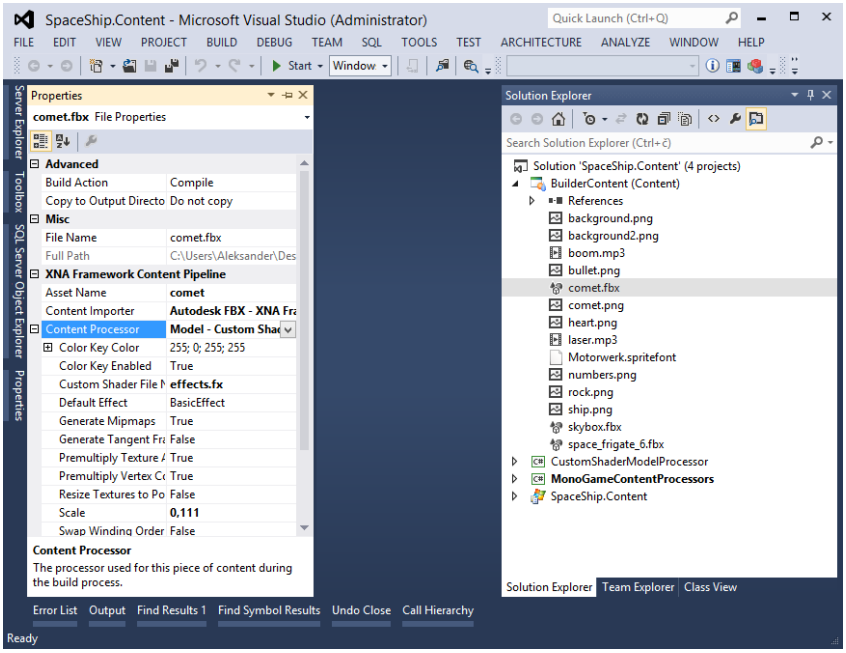
6.2 Cevovod vsebine v ogrodju MonoGame

Ogrodje *MonoGame* je odprtokodna implementacija ogrodja *XNA* in ohranja iste funkcionalnosti kot ogrodje *XNA*. Za *Cevovod vsebine* ogrodja *MonoGame* velja podobno. Tukaj imamo izbiro, da lahko uporabimo *Cevovod vsebine* ogrodja *XNA* ali *Cevovod vsebine* ogrodja *MonoGame*.

Če se odločimo za *Cevovod vsebine* orodja *XNA*, moramo namesto osnovnih priloženih (angl. bulit-in) procesorjev vsebine, uporabiti razširitvene procesorje vsebine ogrodja *MonoGame*. Ti procesorji so razširitev osnovnih procesorjev. Njihova naloga je, da se v primeru, če ciljna mobilna platforma ne omogoča uvoz obdelave vsebine z osnovnim procesorjem, uporabi drugačni procesor, če pa omogoča, se pa uporabijo osnovni procesorji. To je predvsem pri platfromi *iOS* in pri senčilnikih za vse platforme. Vsako vrsto vsebine si bomo ogledali podrobno v naslednji točki.



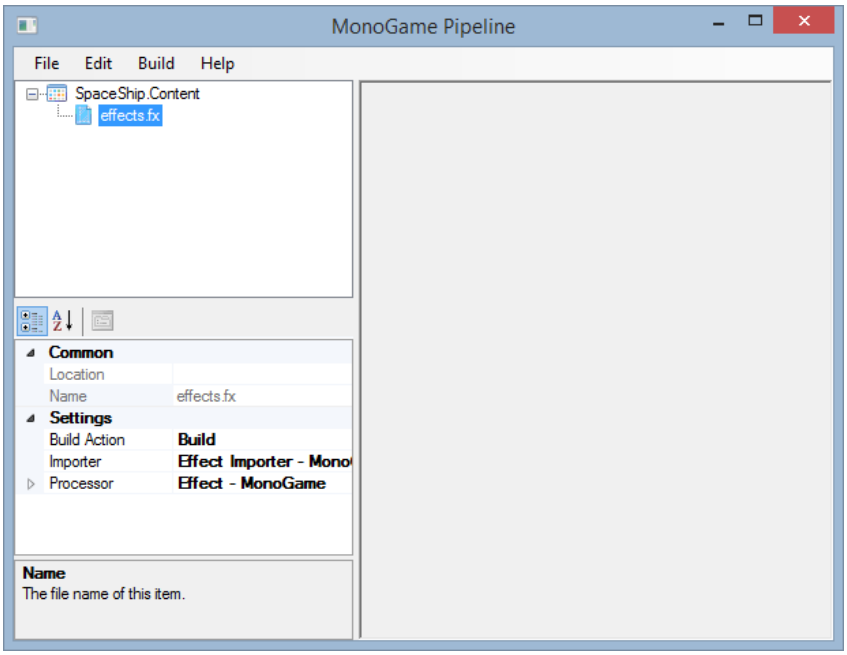
Slika 6.2: Razširitveni procesorji vsebine ogrodja *MonoGame*.



Slika 6.3: *Cevovod vsebine* ogrodja *XNA* v *Visual Studiu 2012*.

Cevovod vsebine ogrodja *MonoGame* je trenutno v razvoju in še ne podpira obdelave vseh vrst vsebine [31]. Ta *Cevovod vsebine* vsebuje en dodatni uvoznik vsebine za video vsebino tipa *H.264* ter ne vsebuje uvoznika in procesorja za vsebino *XACT* projektov. Prednosti tega cevovoda so predvsem, da ne potrebujemo naložiti ogrodja *XNA* in podpira večji nabor ciljnih platform, za katere prevajamo vsebino. Seveda tudi omogoča, da lahko za razvojno okolje uporabimo platformo *Linux* ali *Mac OS X* poleg platforme *Windows*.

Izbira *Cevovoda vsebine* v veliki miri ne vpliva na končni izdelek. Del vsebine lahko prevedemo v enem cevovodu, drugi del v drugem. Oba cevovoda izvozita vsebino v isto obliko in v vmesno datoteko `.xmb`, ki je istega tipa v obeh *Cevovodih vsebine*. Za uporabo *Cevovod vsebine* ogrodja *XNA* moramo ustvariti nov projekt v *Visual Studiu*, *Cevovod vsebine* ogrodja *MonoGame* pa deluje, kot samostojni program. Prednost samega ogrodja *MonoGame*



Slika 6.4: *Cevovod vsebine* ogrodja *MonoGame*.

pred ogrodjem *XNA* je v tem, da omogoča nalaganje neobdelanih enostavnih vsebin, kot so slike tipa `JPG` in `PNG` ter glasba tipa `MP3` [32]. Z obdelavo take vsebine ne pridobimo prednosti s *Cevovodom vsebine* (majhna velikost in hitro nalaganje), ker je ta vsebina že sama po sebi obdelana.

6.3 Vsebina

6.3.1 Model

Ko omenjamo model v igrah, mislimo na 3D modele objektov, ki so predstavljeni s pomočjo poligonov oziroma poligonskih mrež (angl. mesh). Navadno izdelamo modele v namenskih programih za 3D oblikovanje. *Cevovod vsebine* podpira uvoz dveh formatov: *Autodesk FBX* z končnico datoteke `.fbx` in format *DirectX* s končnico `.x`. Pomemben faktor je izbor programa za

izdelavo 3D modelov, saj z vsakim programom ne moremo shraniti modelov v ustrezen format. Večina priljubljenih programov, kot so *3ds Max*, *Maya*, *Blender* in *Wings3D*, omogoča shranjevanje modelov v ustrezne formate.

Modeli so eni bolj ključnih delov naših iger in po navadi poleg samih poligonskih mrež vsebujejo tudi druge vsebine. Vsebujejo še animacije, teksture, senčilnike in vse ostalo kar je lahko neposredno povezano na prikaz modelov. Kako se uvozijo ostale vsebine, bomo obravnavali v naslednjih točkah. Ob obvozu ene vsebine - modela lahko uvozimo verigo vsebin. V tehničnem smislu to pomeni, da procesor modela mora sodelovati tudi z drugimi deli *Cevovoda vsebine*. Tako da je del *Cevovoda vsebine*, ki skrbi za uvoz in obdelavo modelov, najbolj kompleksni del *Cevovoda vsebine*.

Pri *Cevovodu vsebine* ogrodja *XNA* se ločita uvoznika za vsak format posebej `.fbx` in `.x`, pri *Cevovodu vsebine* ogrodja *MonoGame* imamo samo en skupni uvoznik, ki zna uvoziti oba formata.[1] Uvozniki poskrbijo za obdelavo in optimizacijo podatkov iz datoteke modela, ki je prišla iz programa za 3D oblikovanje. Te datoteke po navadi vsebujejo tudi za nas nepomembne, dodatne podatke, ki so namenjeni samo programu, v katerem so bili ustvarjeni. Odstranijo se informacije, ki niso ključne za ogrodje *MonoGame*, odstranijo se odvečna ali podvojena oglišča v mreži modela, pretvorijo se v desni koordinati sistem itd. Glavna naloga je, čim bolj poenostavi opis modela, da ga bomo lažje obdelali v procesorju modela. V obeh *Cevovodih vsebinah* je samo eden procesor modela. Procesor dodatno še obdela vsebino modela za optimalno prikazovanje in požene še druge uvoznike in procesorje, ki so vezani na vsebino modela. Seveda lahko programer napiše razširitev procesorja modela in manipulira z podatki, ki so prišli iz uvoznika modela, ter spremeni končni izgled modela. Končni izdelek je datoteka ali datoteke s končnico `.xmb`, ki jih zna ogrodje *MonoGame* hitro prebrati in naložiti v času izvajanja.

[1] *Cevovodu vsebine* ogrodja *MonoGame* ima posebej uvoznik za format `.fbx`, vendar je samo bližnjica do skupnega uvoznika

6.3.2 Grafični učinek (angl. Effect) – Senčilnik

Grafični učinek je skupek senčilnikov - ogliščnega (angl. vertex) senčilnika in fragmentnaga/točkovnega (angl. fragment/pixel) senčilnika, stanja grafičnega cevovoda (nastavitve Z-bufferja, nastavitev izločanja zadnjih ploskev ipd.) ter tehnik izrisovanja grafičnega cevovoda (katera kombinacija senčilnikov se bo izvedla v grafičnem cevovodu). Grafični učinek (senčilnik) je tekstovna datoteka z končnico `.fx`. Programski jezik senčilnikov ogrodja *MonoGame* je visoko-nivojski jezik senčenja *HLSL* (angl. *High Level Shading Language*), ki je podvrst programskega jezika tipa *C*. Vendar ta programski jezik vsebuje okrnjen nabor vseh ukazov jezika *C*, pretežno samo tiste, ki so namenjeni delu z matrikami in vektorji. Kjub temu, da je programski jezik *HLSL* namenjam samo *Windows* oziroma *DirectX* platformam in da se na platformah *OpenGL* uporablja jezik senčenja *GLSL* (angl. *OpenGL* Shading Language), nam ni potrebno ponovno napisali senčilnikov. Ob procesiranju grafičnih učinkov orodje *MojoShader* v ozadju poskrbi za pretvorbo med jezikoma senčenja, ko je izbrana platforma tipa *OpenGL* [33]. Orodje *MojoShader* pretvori *HLSL* bitno kodo v *GLSL* bitno kodo, ampak tukaj je nekaj omejitev. Poskrbeti moramo, da v jeziku *HLSL* ciljamo verzijo 3.0 modela senčenja (`vs_3_0` in `ps_3_0`). V primeru, da izberemo drugo verzijo, se senčilniki ne bodo prevedli v *GLSL* bitno kodo. Ta bitna koda ustreza *OpenGL ES jeziku senčenja verzije 1.0* (angl. *OpenGL ES Shading Language 1.0*). Prevzeti model senčenja platforme *Windows Phone* je verzije 4.0 nivoja 9.3 (`vs_4_0_level_9_3` in `ps_4_0_level_9_3`)[23]. Seveda lahko uporabimo `#pragmo` direktive prevajalnika, tako kot pri sami programski kodi igre in rešimo problem različnosti modelov senčenja. Uporabimo `#if SM4` za *Windows Phone* in `#else` za ostali dve *OpenGL ES* platformi.

Uvoz in pretvorba grafičnih učinkov je možna z obema *Cevovodoma vsebine*. Vendar ogrodje *MonoGame* ima svojo izvajalno okolje za grafične učinke z imenom *MGFX* (*MonoGame FX*), ki se razlikuje od tistega v ogrodju *XNA*. Zato je potrebno uporabiti razširitveni procesor pri *Cevo-*

vodu vsebine ogrodja *XNA*, ki kompletno preskoči vgrajeni *XNA* procesor grafičnih učinkov in uporabi tistega iz ogrodja *MonoGame*. *MonoGame* procesor grafičnih učinkov za obdelavo uporablja orodje *2MGFX* ter po potrebi tudi *MojoShader* (če je ciljna platforma *OpenGL ES*). *Cevovod vsebine* ogrodja *MonoGame* deluje po istem postopku, saj ima prevzeto uporabo procesorja grafičnih učinkov ogrodja *MonoGame*.

6.3.3 Pisava (angl. Font, SpriteFont)

V ogrodje *MonoGame* lahko uvozimo pisavo na dva načina. Lahko podamo opisno datoteko `XML`, kjer opišemo vrsto, stil, velikost, nabor ter razmik znakov pisave. Nato *Cevovod vsebine* sestavi teksturo vseh znakov in jo stisne v format `DXT`. `DXT` je algoritem za stiskanje tekstur, prednost tega algoritma je, da nam ni potrebno teksture razširiti, ko jo pošljemo v grafično kartico. Za to poskrbi že sama grafična kartica. Druga možnost je, da že podamo teksturo z znaki, ki jo nato *Cevovod vsebine* samo stisne.

Oba cevovoda imata priložene uvoznike in procesorje. Edina razlika je, da moramo uporabiti *MonoGame*-ovo razširitev *Cevovoda vsebine* ogrodja *XNA*. Ta dodatna razširitev poskrbi, če je ciljna platforma *iOS* (*iPhone*), da se tekstura stisni v format `PVRTC`. Strojna oprema na mobilni platformi *iOS* ne omogoča uporabe formata `DXT`.

6.3.4 Tekstura (angl. Texture)

Uvoznika in procesorja obeh *Cevovodov vsebine* imata nalogo pretvoriti slike v teksture. Namen procesorja je spreminjane lastnosti slike z namenom optimizacije teksture: po potrebi razširi teksture na velikost večkratnika števila dve, stisniti teksturo v format `DXT` ali `PVRTC`, predhodno izračuna prosojnost v barvah (angl. premultiplied alpha), predhodno izračuna več verzij tekstur različnih velikosti (angl. mipmapping) ter poskrbeti, da je zapis barve v

zaporedju 32-bitnega števila *RGBA* – vsaka komponenta je 8-bitna oziroma eden bajt.

Bistvene razlike v med cevovodoma ni. Tukaj velja enako kot pri pisavah, da moramo poskrbeti, da se teksture stisnejo v format `PVRTC` pri platfromi *iOS* (*iPhone*), s pomočjo razširitve pri *Cevovodu vsebine* ogrodja *XNA*. *Cevovod vsebine* ogrodja *MonoGame* ima to zmožnost že sam po sebi.

6.3.5 Zvok (angl. Sound) in zvočni učinek (angl. Sound Effect)

Ogrodje *MonoGame* omogoča predvajanje neobdelanih zvočnih posnetkov tipa `MP3` na vseh treh mobilnih platformah ter tudi v formatu `OGG` in `MID` na platformi *Android*. Odločimo se lahko, da zvok uvozimo preko *Cevovoda vsebine*. Tukaj je manjša razlika od pretvorb ostalih vsebin pri *Cevovodu vsebine*. Obdelana vsebine se ne zapiše v vmesno binarno datoteko `.xmb`. Vsebina se prevede v datoteko tipa `.wma` (oba *Cevovoda vsebine* za platformo *Windows Phone*) ali datoteko `.mp3` ali `.mp4` (za ostali dve platformi). V vmesno datoteko se zapiše relativna datotečna pot do zvočnih datoteke `.wma` (ali `.mp3` oz. `.mp4`). Po obdelavi *Cevovoda vsebine* imamo dve datoteki: vmesno datoteko in datoteko z zvočnim zapisom. Uvoznik vsebine podpira formate vrste `MP3`, `WMA` in `WAV`.

Za zvočne učinke orodje *MonoGame* uporablja zvoke tipa `PCM`. S pomočjo katerega koli od obeh *Cevovodov vsebine* lahko pretvorimo zvok v format `PCM` v primeru izbrane višje kvalitete. Za nižjo kvalitete se zvok pretvori v bolj enostavnejši format `ADPCM`.

Razlika med *Cevovodom vsebine* ogordja *XNA* od *Cevovoda vsebine* ogrodja *MonoGame* je, da *Cevovod vsebine* ogrodja *MonoGame* na platformi *iOS* (*iPhone*) ob izbrani nižji kvaliteti izbere format `IMA ADPCM` (sam `ADPCM` ni podprt na platformi *iOS*). V *Cevovodu vsebine* ogrodja *XNA* moram upo-

rabit razširitveni procesor, ki za platformo *iOS* ob nižjo kvaliteti pretvori v format `PCM` z nižjo stopnjo vzorčenje (angl. sample rate).

6.3.6 Video

Predvajanje video vsebin v ogrodju *MonoGame* trenutno ni podprto na vsem platformah. *Android* in *iOS* (*iPhone*) sta trenutno edini dve mobilni platformi, kjer je to mogoče.

Tako kot pri zvoku in teksturah lahko predvajamo video posnetke, ki niso bili obdelani s *Cevovodom vsebine*. Če se odločimo za cevovod, se video vsebina samo skopira v novo datoteko in se ustvari vmesna datoteka, ki vsebuje relativno pot do datoteke z video vsebino. Procesor video vsebine ne obdeluje video vsebine: vhodna datoteka je identična izhodni (postopek je enak kot pri zvoku). Uvozimo lahko video vsebine tipa `WMV` in `H.264`. Slednji tip je možno uvoziti samo na *Cevovodu vsebine* ogrodja *MonoGame*.

6.3.7 XACT Projekt (angl. XACT Project)

V igre ogrodja *MonoGame* trenutno še ne moremo uvoziti glasbo *XACT* projekta.[2] Predvajalnik glasbe *XACT* projektov je še v razvoju in predvajanje te vsebine trenutno še ne da želenega rezultata, ki bi ustrezal enakosti ogrodju *XNA*.

Za uvoz te vrste vsebine lahko uporabimo samo *Cevovod vsebine* ogrodja *XNA*. *Cevovod vsebine* ogrodja *MonoGame* ne vsebuje uvoznikov in procesorjev za *XACT* projekt.

[2]Ob uporabi dnevne razvojne različice ogrodja *MonoGame* je predvajanje glasbe *XACT* projekta delno možna.

Poglavje 7

Izdelava demonstracijske igre

7.1 Kratek opis poteka razvoja iger

V tem poglavju bomo predstavili razvoj dveh iger, tako 2D kot 3D igre, ki je potekal vzporedno z analiziranjem arhitekture in delovanja orodja *MonoGame* na mobilnih platformah.

Pri izdelavi igre smo se najprej odločili za bolj konservativen pristop in v prvi fazi razvili 2D igro. Saj je ogrodje *MonoGame* še le v verziji 3.0 dobilo podporo za razvoj 3D iger in je vsebovalo nekaj pomanjkljivosti in napak. Kasneje smo te pomanjkljivosti tako odpravi, da smo prenesli izvorno kodo iz spletnega repozitorija *GitHub*. Tam se nahajaj izvorna programska koda ogrodja *MonoGame*, kjer se dnevno dodajo popravki. Te popravki se tudi nahajajo v najnovejši verziji z oznako 3.2. Razvoj 2D iger je bolj enostaven in manj zahteven za programsko in strojno opremo mobilnih naprav. V začetku nismo vedeli kakšne zmogljivosti omogoča samo ogrodje *MonoGame*. Tako da smo v prvem koraku preverili delovanje ogrodja z 2D igro.

Za razvoj igre za platformo *iOS* (*iPhone*) se nismo odločili, ker že obe izbrani platformi prestavljata željene funkcionalnosti, ki smo jih želeli preveriti.

To so predvsem:

- različnost v programskem vmesniku za dostop do grafične strojne opreme – *DirectX* in *OpenGL*,
- različnost operacijskih sistemov – *Android* in *Windows Phone* ter
- različnost strojne opreme in različne velikosti grafičnih zaslonov.

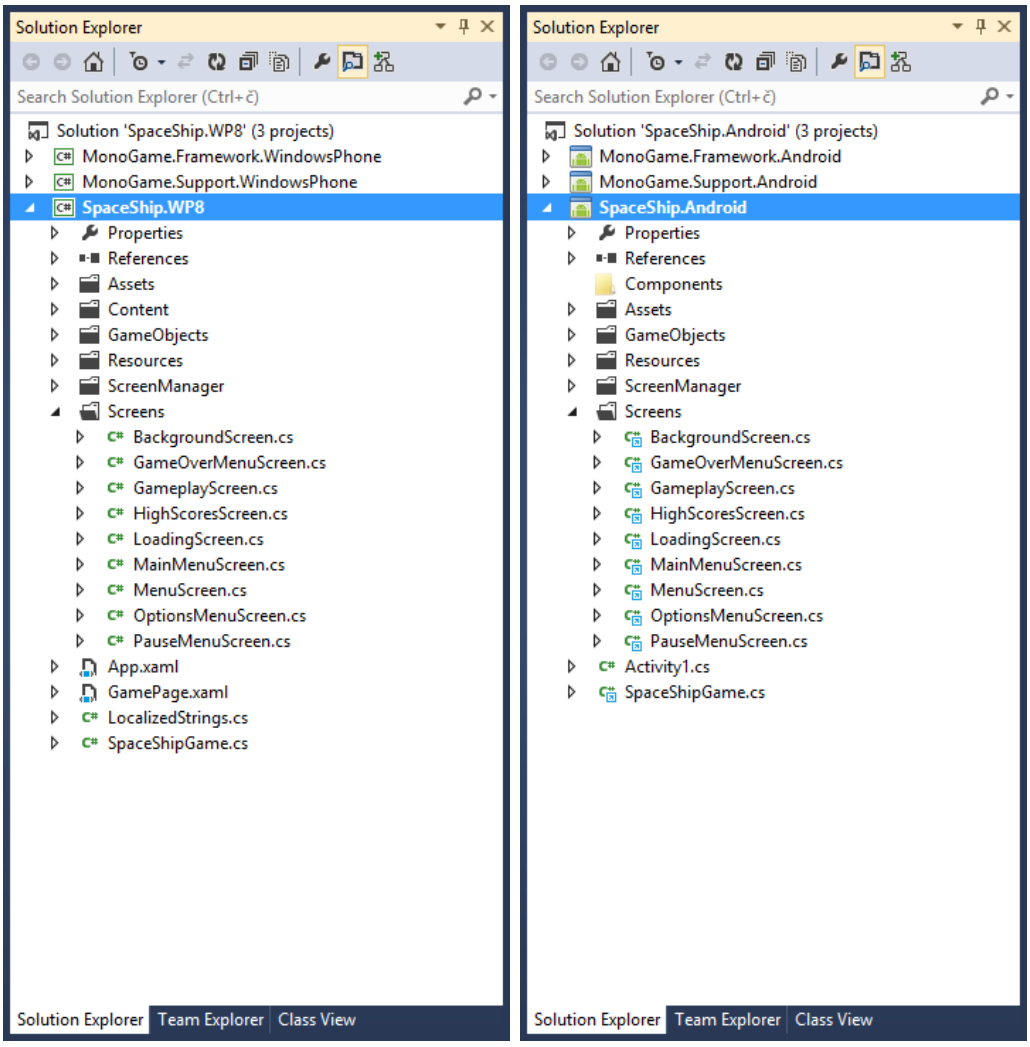
7.2 Opis igre

Izdelani igri sta vesoljski strelski igri (angl. shooter game), z imenoma *Space Ship* in *Space Ship 3D*. V igri vodimo vesoljsko ladjo skozi galaksijo, pri tem se more igralec izogibati meteoritom in jih karseda veliko uničiti. Meteoriti se v 2D igri premikajo iz desne strani proti levi, pri 3D igri iz ozadja proti ospredju. Za vsak uničen meteorit uporabnik prejme pet točk. Meteorit uničimo tako, da ga trikrat zadenemo z izstrelkom, ki potuje iz vesoljske ladje do njega. Izstrelek igralec ustvari s pritiskom na ekran. Vesoljsko ladjo krmilimo po zaslonu s pomočjo nagibanja mobilne naprave. Pri 2D igri lahko ladjo premikamo v vse štiri strani - desno, levo, naprej ter nazaj, pri 3D igri spreminjamo naklon ladje v te štiri strani. Na začetku igre ima igralec tri življenja, ki so prikazana z rdečimi srci v spodnjem desnem robu zaslona. Ob vsakem dotiku vesoljske ladje z meteoritom, se vesoljski ladji moč zmanjša za deset procentov. Ko igralcu pade moč vesoljske ladje na nič, igralec izgubi eno življenje. Začetna moč vesoljske ladje je sto procentov. V primeru, da nimamo več življen in moči, se igra zaključi. Če pa še imamo življenja, se moč vrne na sto, vendar se število življenj zmanjša za eno. Ob zaključku igre se shrani dosežen rezultat na lestvico najboljših rezultatov.

7.3 Vzpostavitev razvojnega okolju

Razvoj igre je potekal z razvojnem okoljem *Visual Studio 2012*. Z namestitvijo ogrodja *MonoGame*, so se v *Visual Studiu* pojavil novi projekti za razvoj iger v ogrodju *MonoGame*. Usvarili smo dva ločena projekta za platfromi *Windows Phone* in *Android*. Poimenovali smo jih `SpaceShip.WP8` in `SpaceShip.Android`. Kasnjene smo za 3D igro ustvari še dva projekta z imenoma `SpaceShip3D.WP8` in `SpaceShip3D.Android`. Za prepoznavo projektov platforme *Android* v *Visual Studiu* potrebujemo še zbirko orodij *Xamarin*. Kljub temu, da sta si oba projekta zelo podobna in vsebujeta datoteke programskega jezika *C#*, razvojno okolje *Visual Studio* potrebuje konfiguracijske datoteke za razpoznavo vrste vsakega projekta. Te konfiguracijske datoteke vsebujejo informacije, kako se mora prevesti izvorna koda v končno obliko (izbira prevajalnika), kako *Visual Studio* komunicira s razhroščevalnikom ipd. Projekti platforme *Windows Phone* so že prevzeto podprti v razvojnem okolju *Visual Studio*.[1] V naslednjem koraku smo s pomočjo odjemalca *GitHub for Windows* prenesli iz spletnega repozitorija *GitHub* zadnjo verzijo izvornih datotek ogrodja *MonoGame*. V mapi, kamor smo prenesli izvorne datoteke, se tudi nahajajo projekti za izdelavo ogrodja *MonoGame*. Projekta `MonoGame.Framework.Android` in `MonoGame.Framework.WindowsPhone` smo vsakega pripeli ustreznima projektoma naše igre `SpaceShip`. Obema projektoma smo morali naprej odstrani referenco na ogrodje *MonoGame*, ki kaže na prevedene datoteke nameščene z ogrodjem *MonoGame*. Te datoteke so vsebovale nekaj napak. Namesto te reference smo naredili novo referenco na pripeti projekti ogrodja *MonoGame*, ki smo ga prenesli iz spletnega repozitorija *GitHub*. S tem smo rešili pereče probleme. Tako smo si pripravili delovno okolje, v katerem smo gradili naše dve igri. Vsa orodja, omenjena v tej točki, so podrobno opisana v *četrtem poglavju*.

[1] Potrebujemo *Windows Phone 8 SDK* vendar se namesti skupaj z *Visual Studiom 2012*



Slika 7.1: Projekta `MonoGame.Framework.WindowsPhone` (levo) in `MonoGame.Framework.Android` (desno) uporabljena v projektu igre.

7.4 Glavna zanka igre ogrodja MonoGame

Preden se posvetimo razvoju igre, si poglejmo delovanje glavne zanke igre orodja *MonoGame*. Glavna zanka igre se izvaja v razredu `Game`. Da lahko dodamo poljubne funkcionalnosti igri, moramo ustvariti novi razred, ki je izpeljan iz (deduje) razreda `Game`. Ta razred smo poimenovali `SpaceShipGame`

in je glavni razred igre. Pred začetkom delovanja zanke imamo konstruktor razreda ter metodi `Initialize` in `LoadContent`. V konstruktorju razreda nastavimo lastnosti igre (hitrost prikazovanja slik v sekundi, orientacija zaslona ...) ter dodamo komponente igre. Te lastnosti in komponente se inicializirajo v metodi `Initialize`. Vsebino, ki smo jo predhodno obdelali v *Cevovodu vsebine*, naložimo v metodi `LoadContent`. Ko se naloži vsebina, se začne neskončno izvajanje glavne zanke igre. Znotraj neskončne zanke se nahajata dve metodi `Update` in `Draw`. Namen metode `Update` je izvajanje logike igre. Metoda `Draw` skrbi za izrisovanje vsebine na grafični zaslon naprave. V primeru počasnega delovanja igre, lahko glavna zanka začasno začne izpuščati klicanje metode `Draw`. S tem dosežemo bolj tekoče delovanje igre na račun manj prikazanih slik v sekundi. Ob izhodu iz igre se pokliče metoda `UnloadContent` ter ustavi se delovanje glavne zanke. Metoda `UnloadContent` odstrani vso vsebino igre, ki smo jo naložili v metodi `LoadContent`.

7.5 Razvoj 2D igre

Za razvoj naše igre smo potrebovali okvir, ki skrbi za upravljane stanj igre, sistem za prehod med različnimi meniji, vodenje lestvice najboljših rezultatov, upravljanje z nastavitvami, nalagalnik igre in njenih objektov, ohranjanje stanja igre, ko igra preide v ozadje (npr. ob telefonskem klicu ali zagonu druge aplikacije). Za ta okvir smo izbrali že narejen brezplačen primerek, ki se nahaja na Microsoftovi strani.[2] Poleg tega, da smo prihranili čas z uporabo tega okvirja, smo tudi delno preverili, če so funkcionalnosti ogrodja *MonoGame* enake ogrodju *XNA*. Kot smo že omenili, je ogrodje *MonoGame* odprtokodna implementacija ogrodja *XNA* in naj ne bi bilo večjih razlik med obema ogrodjema v smislu funkcionalnosti programske kode. Seveda

[2] Dostopno na:
http://xbox.create.msdn.com/en-US/education/catalog/sample/game_state_management

smo okvir delno prilagodili našim potrebam in dodali možnost vodenja lestvice najboljših rezultatov. Razred `ScreenManager` je upravljalec zaslonov naše igre, ki skrbi za prikazovanje različnih zaslonov in ostalih omenjenih funkcionalnosti igre. Igra vsebuje osem različnih zaslonov:

Razreda `BackgroundScreen` predstavlja enostavni zaslon, ki služi kot ozadje vsem menijskim zaslonom. Pri menijskih zaslonih razredov `PauseMenuScreen` in `GameOverMenuScreen` se ta zaslon ne prikaže. V času igranja igre imamo že ozadje in ga nam ni potrebno dodatno prikazovati.

Razred `MainMenuScreen` je začetni zaslon, ki se prikaže igralcu ob zagonu igre. Ta zaslon je menijski zaslon in prikazuje gumbe do drugih zaslonov. Imamo štiri gumbe: Igraj (angl. Play), Lestvica rezultatov (angl. High Scores), Nastavitve (angl. Options) ter Izhod (Exit). Vsi menijski zasloni so izpeljani iz razreda `MenuScreen`.

Razred `OptionsMenuScreen` je menijski zaslon, namenjen prikazovanju nastavitev igre. Vklopimo ali izklopimo lahko vibriranje telefona ob dotiku meteorita z vesoljko ladjo in predvajanje različnih zvokov v času igranja igre.

Razred `HighScoresScreen` je menijski zaslon, na katerem se nahaja lestvica najboljših rezultatov. Lestvica hrani zadnjih deset najboljših rezultatov. Najboljši rezultati so tudi časovno urejeni. V primeru, da sta dva ali več rezultatov istih, je prej doseženi rezultat višje uvrščen na lestvici.

Razred `LoadingScreen` je posebni zaslon, katerega namen ni prikazovanje vsebine uporabniku. Ta zaslon namreč omogoča, da v ozadju naložimo drug zaslon, preden se prikaže. To je takrat uporabljeno, ko zaslon potrebuje nekoliko več časa, da se prikaže. Ta čas uporabniku prikažemo sporočilo ≫*Loading...*≪. Kar obvesti uporabnika, da se bo želenji zaslon kmalu prikazal.

Razred `GameplayScreen` je igralni zaslon, kjer se dejansko izvaja igra. Zaslon skrbi za implementacijo logike igre ter za njeno izrisovanje na grafični zaslon. Igralni zaslon mora naložiti večjo količino vsebine igre, zato smo

uporabili posebni zaslon za nalaganje, omenjen v prejšnjem odstavku. Bolj podrobno delovanje igralnega zaslon si bomo pogledali v naslednjem odseku.

Razred `PauseMenuScreen` je menijski zaslon, ki se prikaže v času premora igre ob pritisku sistemske tipke nazaj (angl. back) za zapustitve zaslona, ali ko aplikacija preide v ozadje. Tukaj ima igralec možnost nadaljevanja igre, ponovnega začetka igre ali izhoda iz igre in vrnitev na začetni zaslon.

Razred `GameOverMenuScreen` predstavlja menijski zaslon ob koncu igre. Igra se konča, ko igralec izgubi vsa tri življenja in moč vesoljske ladje pade na nič procentov ali doseže maksimalno število točk (9999 točk). Ob zaključitvi igre se pogleda število osvojenih točk in se jih primerja z lestvico najboljših rezultatov. Če je število točk večje, kot kateri od rezultatov na lestvici, se trenutno število točk doda v lestvico ter odstrani zadnji rezultat na lestvici. Če se je rezultat uvrstil na lestvico najboljših rezultatov, se igralcu pokaže dialog za vpis imena igralca. Ob potrditvi dialoga se rezultat z imenom shrani na lestvico najboljših rezultatov.

Pomembno lastnost razreda `ScreenManager`, ki jo moramo omeniti, je upravljanje z zasloni ob prehodu igre v ozadje. To se zgodi takrat, ko bodisi sprejmemo telefonski klic ali odpremo drugo aplikacijo. Zaradi omejenih virov na mobilnih napravah in narave mobilnih platform se lahko igra popolnoma odstrani iz pomnilnika, kljub temu da jo nismo popolnoma zaprli. Za zaznavanje prehoda igre v ozadje ogrodje `MonoGame` že ponuja dve metodi. Metodi `OnDeactivated` in `OnActivated` glavnega razreda `SpaceShipGame` se sprožita ob prehodu v ozadje ter ob ponovnem vračanju. Metodi obvestita razred `ScreenManager`, ta poskrbi, da za vsak aktiven zaslon shrani trenutno stanje zaslona. Temu pravimo tudi serializacija. Za vsak zaslon lahko definiramo katere lastnosti lahko shranimo za povrnitev stanja. Zaradi nepredvidljivosti telefonskih klicev, smo se odločili, da vsako spremembo, ki jo naredi uporabnik, takoj shranimo. Tako, da nam ni potrebno shranjevati trenutnega stanja zaslonov. Zapomnimo si samo, da je bil zaslon prikazan. Izjemoma v igralnem zaslonu, kjer poteka igra in se stalno spreminjajo vre-

dnosti, ni smotrno shranjevati sprememb sproti. Tako moramo za igralni zaslon sprožiti serializacijo ob prehodu igre v ozadje. Shranjena stanja, ki so zapisana v datoteke za vsak zaslon posebej, shranimo v lokalni zaščiten pomnilnik telefona, ki je izključno namenjen igri. To omogoča, da uporabnik ne more namerno spreminjati stanj igre (npr. rezultata) ter se datoteke ne morejo pomešati z ostalimi datotekami iger na telefonu. Kaj vse se serializira v igralem zaslonu, si bomo podrobneje pogledali v naslednjem odseku.

Igralni zaslon

Igralni zaslon (razred `GameplayScreen`) predstavlja igralni del igre, zadolžen za izvajanje logike posameznih objektov igre ter njihovo prikazovanje na zaslonu mobilne naprave. Vsi objekti igre so izpeljani iz razreda `GameObject`. Razred `GameObject` hrani lastnosti o objektu, kot so: lokacija objekta na zaslonu, hitrost premikanja po zaslonu, ali je objekt aktiven, kakšno škodo povzroči ob stiku z drugim objektom, moč objekta, širino, višino ter teksturo objekta za prikazovanje na zaslonu. Vsak tak objekt lahko vsebuje notranjo logiko obnašanja objekta (metoda `Update`) ter izrisovanje na zaslonu (metoda `Draw`). V primeru, da objekt ni aktiven (to označuje lastnost `IsAlive`) se omenjeni metodi ne izvajata.

Na zaslonu prikazujemo naslednje objekte:

Razred `Player` predstavlja igralčevo vesoljsko ladjo. Ladjo lahko premikamo v vse štiri smeri zaslona. Premikaje je omejeno na meje zaslona, tako da je vesoljska ladja nikoli ne zapusti meje vidnega polja. Za razliko od ostalih objektov igre, ki sami skrbijo za premikanje in odstranjevanje iz zaslona, za upravljanje tega objekta skrbimo znotraj igralnega zaslona. Premikanje je izvedeno preko nagibanje mobilne naprave. Ob premikanju naprave preberemo spremembo, ki jo zazna pospeškomer (angl. accelerometer) naprave.

Razred `Spacerock` je meteorit, katerega igralec mora uničiti. Sam razred

vsebuje logiko za premikane objekta po zaslonu v metodi `Update`. Meteoriti potujejo z desne strani proti levi strani zaslona. V primeru, ko meteoritu pade moč na nič, zadene vesoljsko ladjo igralca ali zadane levo stran zaslona, se uniči. Vrednost lastnosti `IsAlive` se postavi na `false`. Moč meteorita pade na vrednost nič, če ga trikrat zadenemo z izstrelkom. Za dodajanje meteoritov v igro skrbi igralni zaslon. Maksimalno število meteoritov, ki se lahko istočasno prikažejo je omejeno na število osem. Ko eden od meteoritov postavne neaktiven, ga igralni zaslon ponovno doda v igro. Vsak nov meteorit začne pot skrajno desno, višina začetka poti se izbere naključno v mejah višine zaslona. Vse meteorite hranimo v seznamu `List<Spacerock> spacerocks`. Z metodo `AddSpacerock` dodajamo nove meteorite v igro.

Razred `Bullet` predstavlja izstrelke, s katerim igralec poskuša zadeti meteorite. Izstrelke izstreli igralec ob pritisku na zaslon. Izstrelki se premikajo od trenutne lokacije vesoljske ladje proti desni strani zaslona. Med potjo lahko izstrelek zadene meteorit in mu zmanjša moč za tretjino, izstrelek se uniči in postane neaktiven. Vse izstrelke hranimo v seznamu `List<Bullet> bullets`. V primeru novega pritiska na zaslon, se izbere prvi neaktiven izstrelek iz seznama. V začetku igre napolnimo omejen seznam z neaktivnimi izstrelki. S tem dosežemo hitrejšo prikazovanje izstrelkov ob prvem izstrelu. Razred `Bullet` vsebuje svojo implementacijo metode `Update`, ki skrbi za premikanje izstrelka.

Razred `ScoreBoard` je objekt za prikazovanje števil. Ta objekt smo uporabili za prikazovanje števila trenutno osvojenih točk (štiri-mestno število) ter za prikazovanje trenutne moči vesoljske ladje (tri-mestno število). Oba objekta sta prikazan v zgornjem desnem kotu.

Razred `HealthBoard` je objekt za prikazovanje števila življen igralca. Objekt prikazuje sliko srca za vsako življenje. Ob vsaki izgubi življenja se odstrani eno srce iz prikazovanja. Na začetku igre ima igralec tri življenja, zato so prikazana tri srca.

Razred `Explosion` je animiran objet izpeljan iz razreda `AnimatedGameObject`. Razred `AnimatedGameObject` je posebna vrsta razreda `GameObject`, ki omogoča animacijo. Animacijo dosežemo tako, da teksturo objekta razdelimo na enakomerne sličice. Ob vsakem obhodu metode `Draw` v zanki, se prikaže sosednja sličica. Ob izrisu zadnje sličice, se objekt odstrani iz zaslona. Razred `Explosion`, ki predstavlja eksplozijo v igri, se prikaže ob uničenju meteorita. Prikaže se tudi kot eksplozija vesoljske ladje, ko igralec izgubi vsa življenja ter moč vesoljske ladje pade na vrednost nič. Vse eksplozije hranimo v seznamu `List<Explosion> explosions`, ki jih ob ponovnem prikazovanju uporabimo že eno od neaktivnih eksplozij, tako kot pri izstrelkih in meteoritih.

Igralni zaslon skrbi za zaznavanje trkov med objekti. Metoda `UpdateCollisions` v enojni zanki zaznava trke vesoljske ladje. V metodi `UpdateBullets` s pomočjo dvojne zanke preverjamo trke izstrelkov z meteoriti. Trk med objekti se zgodi, ko se površine objektov pokrivajo. Kljub temu, da imamo dvojno zanko v metodi `UpdateBullets`, je zaznavanje trkov zelo hitro.

Zdaj, ko smo definirali vse objekte igre, si lahko pogledamo, kaj vse moramo shraniti v primeru nenadnega prehoda igre v ozadje. Za vsak omenjen objekt v igralnem zaslonu moramo ohraniti stanje ob ponovni vrnitvi v igro. V metodi `Serialize` shranimo trenutni rezultat, število življenj, moč vesoljske ladje, ter sezname meteoritov, izstrelkov in eksplozij. Ob povratku v igro z metodo `Deserialize` povrnemo stanje igre. Posrbeti moramo, da vračanje v igro ne poteka predolgo. Za shranjevanje moramo izbrati samo tiste dele (stanja) igre, ki so nujno potrebni.

7.5.1 Medplatofrmonost igre

Pomembni faktor pri medplatformnih igrah je, koliko programske kode je skupne med različnimi platformami. Projekta `SpaceShip.WP8` in

`SpaceShip.Android` si delita iste izvorne datoteke. V projektu `SpaceShip.WP8` smo najprej ustvarili izvorne datoteke, ki smo ji kasneje povezali v projekt `SpaceShip.Android`. Namesto, da prekopiramo datoteke v drugi projekt, izberemo možnost – `Add as Link` (slov. dodaj kot povezavo) in s tem povežemo datoteke v drugi projekt. To pomeni, da lahko izvorne datoteke urejamo samo enkrat in spremembe bodo vidne, tako v projektu za platformo *Windows Phone* kot v projektu za platformo *Android*. To je velika prednost, ki nam omogoča razvijanje igre za več platform istočasno in uporabo istih izvornih datotek. Vse to nam olajša preverjanje in razhroščevanje izvorne kode.

Seveda vse izvorne kode ni bilo mogoče deliti med obema platformama. Ustvarili smo ločeni projekt `MonoGame.Support`, kjer smo za vsako platformo posebej ustvarili programske datoteke za reševanje različnosti platform. Največ težav smo imeli z imenskim prostorom `Microsoft.Devices`. Ta imenski prostor je del platforme *Windows Phone* in je tudi delno podprt v ogrodju *MonoGame* za ostale platforme. Vendar je v ogrodju *MonoGame* manjkala podpora za razred `VibrateController`, zato smo morali sami napisati implementacijo tega razreda. Na platformi *Windows Phone* nam je pospeškomer povzročal preglavice. Razred `Accelerometer` v imenskem prostoru `Microsoft.Devices.Sensors` vrača ob branju trenutnega razreda `Vector3`, ki je del ogrodja *XNA*. Do zapleta je prišlo, ker ogrodje *MonoGame* vsebuje imensko isti razred v istem imenskem prostoru. Ogrodje *MonoGame* uporablja iste imenske prostore kot ogrodju *XNA*. Problem smo rešili z zunanjo referenco na obe ogrodij ter s preimenovanjem obeh razredov `Vector3`. Prikazano na spodnjem Primeru kode 7.1:

```
#if WINDOWS_PHONE
extern alias MicrosoftXnaFramework;
extern alias MonoGameXnaFramework;
using MsXna_Vector3 =
    MicrosoftXnaFramework::Microsoft.Xna.Framework.Vector3;
```

```
using MGXna_Vector3 =
    MonoGameXnaFramework::Microsoft.Xna.Framework.Vector3;
#endif
```

Primer kode 7.1: Zunanja referenca na ogrodij *XNA* in *MonoGame* ter preimenovanje obeh razredov `Vector3`.

Druga težava, ki smo jo našli v ogrodju *MonoGame*, je bila pomanjkljiva implementacija razreda `Guide` pri platformi *Android*. Naloga razreda `Guide` je prikazovanje dialoga za vnos besedila in dialoga za prikaz obvestil. Implementacijo tega razreda za platformo *Android* smo popravili. Kasneje smo tudi popravek objavili na spletnem repozitorju *GitHub*.

Razred `Guide` se nahaja v ločeni knjižnici oziroma projektu `MonoGame.Framework.Net`. Vendar je ta knjižnica odvisna od mrežne knjižnice `Lidgren.Network`. Ker ne potrebujemo omrežne povezave pri igri, nam dodatna knjižnica nepotrebno povečuje velikost končne igre. Prekopirali smo izvorno datoteko za obe platformi v projekt `MonoGame.Support` in odstranili vse povezave na dodatno knjižnico.

7.5.2 Vsebina igre

Vsebino smo obdelali s pomočjo *Cevovoda vsebine* ogrodja *XNA*. Za delovanje *Cevovoda vsebine* smo morali najprej ustvariti projekt ogrodja *XNA* v *Visual Studiu*. Projekt smo poimenovali `SpaceShip.Content`. Nato smo v projekt *Cevovoda vsebine* dodali željeno vsebino. *Cevovod vsebine* sam izbere vrsto procesorja vsebine glede na vrsto vsebine, vendar smo morali izbrane procesorje vsebine zamenjali z razširitvenimi procesorji vsebine ogrodja *MonoGame*. Kot smo omenili v poglavju o *Cevovodu vsebine*, razširitveni procesorji vsebine ogrodja *MonoGame* poskrbijo, da se vsebina pravilno prevede glede na določeno ciljno platformo. Vrste vsebine, ki smo jih uporabili v igri, so: *tekstura*, *pisava* in *zvočni učinek*.

Poskušali smo tudi uporabiti *Cevovod vsebine* ogrodja *MonoGame*, vendar je bil ob prvi uporabi še v zgodnji fazi razvoja in ni vseboval vseh potrebnih uvoznikov in procesorjev vsebine za razvoj igre. Tako, da smo se raje čez cel razvoj igre držali *Cevovoda vsebine* ogrodja *XNA*.

Po obdelavi vsebine s *Cevovodom vsebine* je bilo potrebno prekopirati vse binarne datoteko s končnico `.xmb` v projekt igre. Lokacija in način dodajanja vsebine se med platformam *Windows Phone* in *Android* malo razlikuje.

Pri platformi *Windows Phone* je bilo potrebno prekopirati datoteke v imenik `Content` projekta igre. Vsaki skopirani datoteki vsebine smo morali določiti način prevajanja – `Build Action` v `Content` (slov. vsebina) ter način kopiranja – `Copy to Output Directory` v `Copy If Newer` (slov. kopiraj, če je novejše). S tem smo dosegli, da se je vsebina prenesla v končno igro ob prevajanju izvorne kode in ob zamenjavi vsebine.

Vsebina v projektu platforme *Android* se tudi nahaja v imeniku `Content`, vendar je ta imenik podimenik imenika s sredstvi `Assets`, ki se nahaja v projektu igre. Tukaj nam ni bilo potrebno spreminjati nastavitev datotek vsebine.

7.5.3 Ločljivosti zaslonov mobilnih naprav

Eden od problemov, ki smo ga morali rešiti pri razvoju 2D igre, je upravljanje z različnimi ločljivostmi zaslonov. Poleg različnih velikosti zaslonov, nam preglavice povzročajo različna razmerja med stranicami zaslona. Pri različnih velikostih zaslona, pri istem razmerju stranic, sliko samo raztegnemo ali skrčimo za neko konstanto (faktor), tako da ustreza drugi velikosti zaslona. Če se dva zaslona razlikujeta v razmerju stranic oziroma v slikovnem razmerju (angl. aspect ratio), imamo nekoliko težjo nalogo. V tem primeru enostavno ne moremo narediti isto in sliko raztegniti ali skrčiti za neko konstanto. Pri tem se del slike lahko ne prikaže oziroma odreže iz zaslona ali

pojavi se črna polja na stranicah zaslona. To je prikazano na Sliki 7.2.

Slika 7.2: Ob uporabi "Letterbox" rešitve, se ob stranicah zaslona pojavijo črna polja.

Obstaja več načinov za rešitev tega problema. Način, da prikažemo na stranicah zaslona črna polja imenujemo "Letterbox". Vso vsebino igre ohranimo na zaslona, vendar dajemo igralcu občutek, da igra ni popolnoma namenjena njegovi mobilni platformi. Uporabili smo drugačen pristop, pri katerem smo poskrbeli, da se slika igre vedno prikaže čez celotni zaslon.

V našem pristopu smo uporabili naravno ločljivost zaslona. Naravna ločljivost je tista ločljivost, ki jo določa strojna oprema oziroma zaslon mobilne naprave. Našo igro prilagajamo zaslonu in ne obratno, kot je pri načinu "Letterbox". Najprej si moramo izbrati referenčno ali virtualno ločljivost zaslona. Potrebujemo jo za izdelavo tekstur. Izbrali smo referenčno ločljivost 1280 pikslov v širino in 720 pikslov v višino. Ta ločljivost je bila izbrana na podlagi najpogostejših razširjenih ločljivosti zaslona na mobilnih napravah [34]. Seveda ločljivost 1280x720 ni najbolj pogosta ločljivosti, ampak je v sredini med najvišjimi in najnižjimi najbolj pogostimi ločljivostmi zaslonov [34]. To pomeni, da se bo na večjih zaslonih (1920x1080) tekstura raztegnila,

na manjših zasloni (800x480) pa skrčila, vendar za približno isti faktor. Če moramo teksture preveč raztegniti, se kvaliteta teksture zelo poslabša. Če pa skrčimo za velik faktor, ne poslabšamo kvalitete teksture, vendar v pomnilniku teksture zasedamo več prostora, kot bi potrebovali. V pomnilniku imamo zapisano večjo število pikslov, kot jih potrebujemo.

Najprej moramo izračunati faktor skaliranja, ki pove za koliko moramo raztegniti ali skrčiti objekte v igri. Faktor skaliranja se izračuna, tako da višino naravne velikosti zaslona delimo z višino virtualne. Virtualno višino definiramo v konstruktorju razreda `ScreenManager` ter naravno preberemo iz lastnosti `Height` v razredu `Viewport`, ki je del ogrodja *MonoGame*.

```
public ScreenManager(Game game, int virtualWidth = 1280, int
    virtualHeight = 720) : base(game)
{
    //...
}

public override void Initialize()
{
    scale = this.GraphicsDevice.Viewport.Height /
        (float)virtualHeight;
}
```

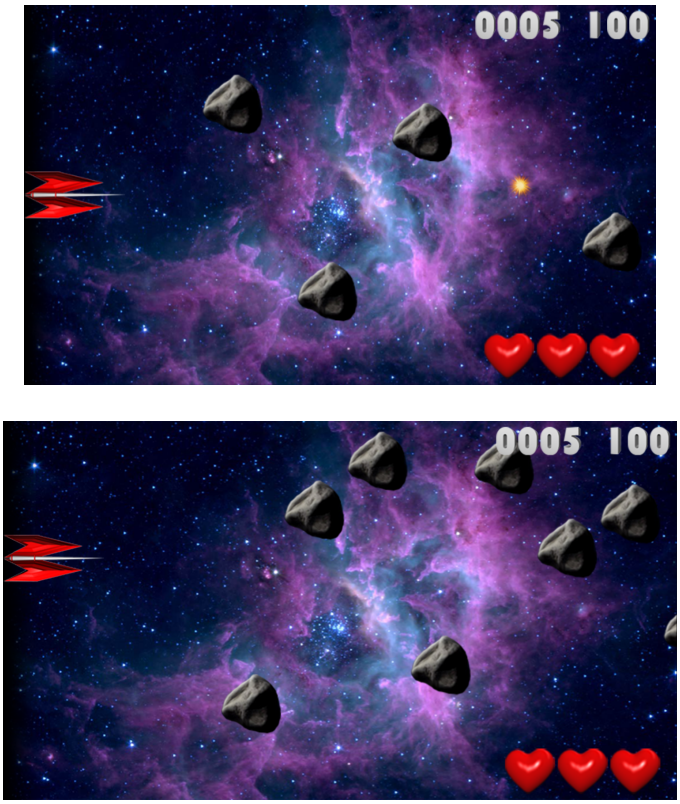
Primer kode 7.2: Izračun faktorja skaliranja in določitev virtualne ločljivosti zaslona v razredu `ScreenManager`.

Ko imamo izračunan faktor skaliranja, ga lahko uporabimo za spreminjanje velikosti objektov v igri. Vsak objekt, ki mu moramo spremeniti velikost, mu moramo podati v konstruktorju argument, ki določat faktor skaliranja. Do faktorja skaliranja lahko dostopamo kjerkoli v igri, tako da preberemo lastnost `ScreenManager.Scale`.

```
scale = ScreenManager.Scale;
player = new Player(PlayerIndex.One, ref playerTexture, scale);
```

Primer kode 7.3: Inicializacija razreda `Player`, ki mu v konstruktorju podamo faktor skaliranja.

Ko smo že omenili, igro prilagajamo vedno naravni velikosti zaslona. To pomeni, da je gibanje objektov igri vedno v mejah velikosti zaslona, objektom samo spremenimo velikost, da niso prikazani preveliki ali premajhni. Pri širokih zaslonih (angl. widescreen) meteoriti prepotujejo daljšo pot, vendar to ne vpliva na igranje (angl. gameplay) igre.



Slika 7.3: Pri širokih zaslonih (spodnji zaslon) meteoriti prepotujejo daljšo pot kot pri ožjih zaslonih (zgornji zaslon).

7.6 Razvoj 3D igre

Pri razvoju 3D igre, smo uporabili večino delov 2D igre. Spremeniti smo morali igralni zaslon (razred `GameplayScreen`) ter dodati 3D objekte. Dodali smo tudi novo vsebino, ki zdaj vsebuje tudi 3D modele in senčilnike.

Igralni zaslon

Logika igre ostaja enaka v 3D igri, spremenil se je le pogled iz 2D sveta v 3D svet. Za upravljanje pogleda smo ustvarili razred `Camera`, ki predstavlja navidezno kamero igre. Razred `Camera` skrbi za določanje smeri pogleda zaslona na 3D svet igre, določanje zornega kota pogleda, slikovna razmerje, prednjo in zadnjo ravnino rezanja. Vsi ti parametri so potrebi za upodabljane slike pri 3D igrah. Ob premikanju navidezne kamere premikamo tudi vesoljsko ladjo.

Slika 7.4: Igralni zaslon v 3D igri.

Naslednjo stvar, ki smo jo morali spremeniti, so bili objekti igre. Objekte igre smo izpeljali iz razreda `Game3DObject`. Razred `Game3DObject` je ena-

kovreden razredu `GameObject` v 2D igri. Oba razreda služita za hranjenje lastnosti, izvajanje logike ter upodabljanje objektov. Glavna razlika je, da za prikazovanja objekta uporabljamo 3D modele namesto tekstur. Razred `Game3DObject` je izpeljan iz razreda `SimpleGame3DObject`. Razlika med tema dvema razredoma je, da razred `SimpleGame3DObject` ne vsebuje 3D modela (razred `Model`). Uporabljamo ga pri objektu razreda `Bullet`, kjer objekt ne izrišemo s pomočjo 3D modela, ampak programsko. Za premikanje in prikazovanju objektov v 3D svetu, smo morali zamenjati vse pojavitve razreda `Vector2` z razredom `Vector3`.

Objekte, ki smo jih spremenili za 3D igro:

Razred `Player` je zdaj izpeljan iz razreda `Game3DObject`. Vesoljsko ladjo prikažemo s 3D modelom, namesto s teksturo. Vesoljsko ladjo premikamo po zaslonu v vse štiri smeri zaslona, vendar smo dodali še nagibanje ladje ob premikanju. Primikanje ladje določa navidezna kamera razreda `Camera`. Premikanje je izvedeno preko nagibanja mobilne naprave. Ob premiku naprave isto časno premaknemo navidezno kamero in vesoljsko ladjo.

Razred `Spacerock` je izpeljan iz razreda `Game3DObject`. Meteoriti se v 3D igri premikajo iz ozadja proti ospredju. Namesto da se premikajo po `X` koordinati, se zdaj premikajo po `Z` koordinati proti ospredju.

Razred `Bullet` je izpeljan iz razreda `SimpleGame3DObject`. Izstrelke izrišemo s pomočjo dveh trikotnikov, ki jih definiramo programsko. S pomočjo senčilnika izstrelke vedno izrišemo tako, da so obrnjeni proti kameri. S tem dosežemo, da predstavimo enostavni 2D objekt kot 3D objekt.

Razreda `ScoreBoard`, `HealthBoard` sta še vedno izpeljana iz razreda `GameObject` in objekta ohranjata še vedno iste funkcionalnosti kot v 2D igri.

Posodobiti smo morali metodi `UpdateCollisions` in `UpdateBullets` za zaznavanje trkov med objekti. Logika obeh metod ostaja ista, vendar namesto primerjanja prekrivanja površin objekt ob trku, zdaj preverjamo prekri-

vanje očrtanih sfer 3D objektov. Podatek velikosti očrtane sfere 3D objekta dobimo iz modela objekta. Sam model ne vsebuje velikosti očrtane sfere, vendar je ta informacija zapisana v mrežah, ki sestavljajo model. Napisali smo metodo `CreateMergedFromModel`, ki izračuna očrtano sfero, tako da združimo vse očrtane sfere mrež modela v eno skupno očrtano sfero.

```
public static BoundingSphere CreateMergedFromModel(this
    BoundingSphere sphere, Model model)
{
    BoundingSphere boundingSphere = new BoundingSphere();

    Matrix[] transforms = new Matrix[model.Bones.Count];
    model.CopyAbsoluteBoneTransformsTo(transforms);

    foreach (ModelMesh mesh in model.Meshes)
    {
        boundingSphere =
            BoundingSphere.CreateMerged(boundingSphere,
            mesh.BoundingSphere);
        boundingSphere.Transform(transforms[mesh.ParentBone.Index]);
    }

    return boundingSphere;
}
```

Primer kode 7.4: Metoda `CreateMergedFromModel`, ki izračuna združeno očrtano sfero modela.

7.6.1 Medplatformnost igre

Pri 3D igre nismo naleti na nobene dodate težave, zaradi različnosti med platformama *Android* in *Windows Phone*. Uporabili smo iste rešitve, kot pri

2D igri. Razlika med platformama se je edino pokazala v senčilnikih. Rešitev tega problema si bomo ogledali v naslednjem odstavku o Vsebini igre.

7.6.2 Vsebina igre

Projektu `SpaceShip.Content` *Cevovoda vsebine* ogrodja *XNA* smo dodali novo vsebino. V 3D igri smo poleg obstoječe vsebine igre uporabili še vsebino vrste: *model* in *grafični učinek – senčilnik*.

Za prikazovanje 3D modelov v igri smo uporabili poljubne grafične učinke oziroma senčilnike. Pri uporabi poljubnih senčilnikov moramo v *Cevovodu vsebine* eksplicitno določiti povezavo med poljubnim senčilnikom in modelom. Napisali smo razširitveni procesor za poljubne senčilnike modela (razred `CustomShaderModelProcessor`), ki je izpeljan iz procesorja modela ogrodja *XNA* (razred `ModelProcessor`). Razširitveni procesor zamenja vgrajene senčilnike s poljubnim ob obdelavi modela. Razširitvenemu procesorju moramo podati ime datoteke senčilnika, ki se mora nahajati v istem imeniku kot model.

Pri senčilnikih, kot smo opisali v točki 6.3.2, moramo poskrbeti, da se prevede za vsako platformo posebej. Za prevajanje smo uporabili razširitveni procesor ogrodja *MonoGame* za ogrodje *XNA*, ki je poskrbel za izdelavo dveh različnih senčilnikov. V kodi senčilnika smo uporabili `#pragmo` direktive prevajalnika, s pomočjo katere lahko ciljamo pravo verzijo modela senčenja znotraj iste datoteke senčilnika.

```
technique Technique
{
   pass Pass1
   {
#if SM4
      VertexShader = compile vs_4_0_level_9_3 VertexShaderFunc();
```

```
        PixelShader = compile ps_4_0_level_9_3 PixelShaderFunc();
#else
        VertexShader = compile vs_3_0 VertexShaderFunc();
        PixelShader = compile ps_3_0 PixelShaderFunc();
#endif
    }
}
```

Primer kode 7.5: Ciljanje dveh različnih platform (modelov senčenja) s pomočjo `#pragme` direktive prevajalnika znotraj iste datoteke grafičnega učinka – senčilnika.

7.6.3 Ločljivosti zaslonov mobilnih naprav

V 3D igri za prikaz slike na zaslonu mobilne naprave uporabljajo perspektivno projekcijo. Pri izrisu perspektivne projekcije podamo slikovno razmerje stranic zaslona in ogrodje *MonoGame* samo poskrbi za pravilno prikazovanje slike igre.

Poglavje 8

Sklepne ugotovitve

Kot smo spoznali v diplomski nalogi, ogrodje *MonoGame* poenostavi razvoj medplatformnih iger za mobilne platforme. Namesto da bi morali ponovno napisati igro za vsako platformo posebej, smo napisali igro znotraj ene izvorne programske kode za vse mobilne platforme. Del programske kode se je še vedno razlikoval med platformami, ampak to za zelo majhen delež. Kljub temu smo še vedno premagovali različnosti platform znotraj istega programskega jezika, istega kot smo ga uporabili za razvoj igre. Spoznali smo tudi nekaj tehnik, uporabljenih tudi pri implementaciji ogrodja *MonoGame*, ki so nam pomagale premagovati različnosti platform. Vse to je nam omogočilo, da nismo potrebovali veliko odstopati od glavnega dela razvoja igre. Zelo pomembna prednost ogrodja *MonoGame*, seveda če odmislimo naše raziskovanje arhitekture ogrodja, da razvijalec igre ne rabi poznati specifičnosti vsake platforme in se lahko osredotoči samo na razvoj igre.

Ugotovili smo, da uporaba ogrodja *MonoGame* omogoča razvoj medplatformnih iger, tudi za mobilne naprave. Spoznali smo tudi, da samo ogrodje *MonoGame* ne rešuje problema različnosti zaslonov mobilnih naprav in mora za to poskrbeti razvijalec sam. Ogrodje *MonoGame* uporablja dva različna programska vmesnika za dostop do grafične strojne opreme. To sta vmesnika

DirectX (za platformo *Windows Phone*) in *OpenGL* (za platformi *Android* in *iOS*). Nekaj pospešitve pri zagonu in izvajanju igre dosežemo z uporabo vsebine, obdelane s pomočjo *Cevovoda vsebine*.

Seveda nobeno orodje za razvoj iger ni popolno. Enako velja tudi za ogrodje *MonoGame*. Med razvojem iger smo se srečevali z različnimi težavami, posebej na začetku pri razvoju 3D igre. Modeli in ostala vsebina se niso pravilno prikazovali, vrednosti se niso pravilno izračunale ipd. Večina teh težav so razvijalci ogrodja *MonoGame* odpravili med našim razvojem iger. Nekatere pomanjkljivosti ogrodja smo tudi samo dopolnili. Tako, da ob koncu razvoja iger nismo imeli več težav. Ogrodje *MonoGame* je še vedno v delnem razvoju, kar pomeni, da lahko naletimo še na kakšno pomanjkljivost, vendar bo tudi ta odpravljena.

Tudi nekaj izboljšav bi lahko dodali našim dvema igrama. Seveda ne smemo pozabiti, da smo izdelali samo demonstracijski igri, s katerima smo poskušali razumeti delovanje ogrodja *MonoGame* in razvoj medplatformnih iger. Veliko izboljšav bi lahko naredi na igralnosti igre. Dodali bi lahko več vrst objektov, ki se premikajo po zaslonu poleg samih meteoritov, takšni ki prinašajo različno število točk ob uničenju, ali če jih poberemo z vesoljsko ladjo, igralcu povečajo moč ali število življenj. Dodali bi lahko še različne nivoje težavnosti. Pri obeh igrah je še kar nekaj prostora za izboljšave. Premikanje vesoljske ladje s pospeškomerom tudi še ni idealno. Potrebovali bi še nekaj časa, da bi ujeli bolj naravno premikanje ladje ob nagibanju mobilne naprave. Ustvarili bil lahko tudi še projekt za platformo *iOS* in spoznali še kakšno novo platformno specifičnost, ki se ni pojavila pri platformah *Android* in *Windows Phone*.

Prihodnost ogrodja *MonoGame* bo uspešna, če bodo razvijalci nadaljevali zdajšnje uspešno delo. Trenutno je največ dela posvečenega razvoju *Cevovoda vsebine*, ki ga na žalost nismo mogli uporabi pri razvoju iger, ker še ni dokončan. Dokončanje *Cevovoda vsebine*, bo prineslo popolno neodvisnost od ogrodja *XNA*, ter omogočilo razvoj iger še na platformah *Linux* in *Mac OS*

X. Ogrodje *MonoGame* cilja na enakost funkcionalnosti programskega vmesnika ogrodja *XNA*. Vendar zadnja verzija ogrodja *XNA* je bila izdana leta 2011 in v svetu računalniške grafike se je že od takrat kar nekaj stvari spremenilo. Nimamo več samo dveh senčilnikov, ampak kar cel nabor različnih senčilnikov. Mobilne naprave so se tudi zelo spremenile v zadnjih treh letih, dobile so kar nekaj novih senzorjev, poleg pospeškomera imamo tudi žiroskop. Vse te novosti bodo morali razvijalci ogrodja *MonoGame* dodati v prihodnjih verzijah ogrodja, če bodo hoteli slediti vsem trendom razvoja medplatformnih iger.

Literatura

[1] (2014) libGDX, "libGDX Goals and Features". Dostopno na: http://libgdx.badlogicgames.com/features.html

[2] (2014) Badlogic Games , "First RoboVM/libgdx app in iOS App Store". Dostopno na: http://www.badlogicgames.com/wordpress/?p=3193

[3] (2014) Google Code, "PlayN". Dostopno na: https://code.google.com/p/playn/

[4] (2014) OGRE – Open Source 3D Graphics Engine, "Features". Dostopno na: http://www.ogre3d.org/about/features

[5] (2014) Unity, "Multiplatform - Publish your game to over 10 platforms". Dostopno na: http://unity3d.com/unity/multiplatform

[6] (2014) Unity, "Scripting - Script your gameplay in a world-leading programming environment". Dostopno na: https://unity3d.com/unity/workflow/scripting

[7] (2014)Unity, "iOS and Android mobile game development". Dostopno na: http://unity3d.com/unity/multiplatform/mobile

[8] (2014) CodePlex, "ANX.Framework". Dostopno na: http://anxframework.codeplex.com/

[9] (2014) M. Jan, "XNI – XNA for iOS - Introduction". Dostopno na: http://xni.retronator.com/post/2850676376/introduction

[10] (2014) CVG, "Microsoft email confirms plan to cease XNA support". Dostopno na: http://www.computerandvideogames.com/389018/microsoft-email-confirms-plan-to-cease-xna-support/

[11] (2014) MonoGame, "About". Dostopno na: http://www.monogame.net/about/

[12] (2014) Gamasutra, "It's official: XNA is dead". Dostopno na: http://www.gamasutra.com/view/news/185894/Its_official_XNA_is_dead.php

[13] (2014) GitHub, "MonoGame". Dostopno na: https://github.com/mono/MonoGame

[14] (2014) Microsoft, "Visual Studio, Application Development". Dostopno na: http://www.visualstudio.com/explore/application-development-vs

[15] (2014) Microsoft, "Visual Studio Express". Dostopno na: http://www.visualstudio.com/products/visual-studio-express-vs

[16] (2014) Xamarin, "Mobile Application Development to Build Apps in C#". Dostopno na: http://xamarin.com/platform

[17] (2014) MonoGame, "Documentation, Class Library Reference". Dostopno na: http://www.monogame.net/documentation/?page=api

[18] (2014) Microsoft, "XNA Framework Class Library". Dostopno na: http://msdn.microsoft.com/en-us/library/bb203940(v=xnagamestudio.40).aspx

[19] (2014) C. Hardy, "Introduction to MonoTouch and Monodroid/Mono for Android". Dostopno na: http://www.slideshare.net/chrisntr/introduction-to-monotouch-and-monodroidmono-for-android

[20] (2014) SharpDX, "SharpDX - Managed DirectX". Dostopno na: http://sharpdx.org/

[21] (2014) Microsoft, "Monogame at Build 2012". Dostopno na: http://channel9.msdn.com/Events/Ch9Live/Channel-9-Live-at-BUILD-2012/Monogame-at-Build-2012

[22] (2014) Microsoft, "Differences in game development between the phone and the desktop for Windows Phone 8". Dostopno na: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662930(v=vs.105).aspx

[23] (2014) Microsoft, "Direct3D feature level 9_3 for Windows Phone 8". Dostopno na: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj714085(v=vs.105).aspx

[24] (2014) Microsoft, "Specifying Compiler Targets," *Direct3D 9.1, 9.2, and 9.3 feature levels* Dostopno na: http://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx#direct3d_9.1__9.2__and_9.3_feature_levels

[25] (2014) OpenTK, "The Open Toolkit Library". Dostopno na: http://www.opentk.com/

[26] (2014) Khronos Group, "OpenGL ES 2.X - for Programmable Hardware". Dostopno na:
https://www.khronos.org/opengles/2_X/

[27] (2014) Android Developers, "OpenGL ES". Dostopno na:
http://developer.android.com/guide/topics/graphics/opengl.html

[28] (2014) Apple, "Checklist for Building OpenGL ES Apps for iOS". Dostopno na:
https://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLESontheiPhone/OpenGLESontheiPhone.htm

[29] (2014) Google Code, "Lidgren networking library generation 3". Dostopno na:
https://code.google.com/p/lidgren-network-gen3/

[30] (2014) S. Hargreaves, "Content Pipeline assemblies". Dostopno na:
http://i1.blogs.msdn.com/b/shawnhar/archive/2008/11/24/content-pipeline-assemblies.aspx

[31] (2014) MonoGame, "Content Build Pipeline". Dostopno na:
https://github.com/mono/MonoGame/wiki/Content-Build-Pipeline/

[32] (2014) N. Danson, "F# and Monogame Part 4 – Content Pipeline". Dostopno na:
http://neildanson.wordpress.com/2013/08/13/f-and-monogame-part-4-content-pipeline/

[33] (2014) MonoGame, "Documentation, Custom Effects". Dostopno na:
http://www.monogame.net/documentation/?page=Custom_Effects

[34] (2014) MarketingProfs, "Mobile - Mobile Trends: Most Popular Phones, Screen Sizes, and Resolutions". Dostopno na:
http://www.marketingprofs.com/charts/2014/25740/mobile-trends-most-popular-phones-screen-sizes-and-resolutions